

6

Control Structures

Control structures—*conditionals* and *loops*—are a staple of programming languages. PHP has two conditionals—**if** and **switch**—both of which you'll master in this chapter. Conditionals allow you to establish a test and then perform actions based on the results. This functionality provides the ability to make Web sites even more dynamic.

The discussion of **if** conditionals requires introduction of two last categories of operators: *comparison* and *logical* (you've already seen the arithmetic and assignment operators in the previous chapters). You'll commonly use these operators in your conditionals, along with the Boolean concepts of *TRUE* and *FALSE*.

Finally, this chapter introduces loops, which allow you to repeat an action for a specified number of iterations. Loops can save you programming time and help you get the most functionality out of arrays, as you'll see in the next chapter.

In This Chapter

Creating the HTML Form	116
The if Conditional	119
Validation Functions	122
Using else	126
More Operators	129
Using elseif	138
The Switch Conditional	142
The for Loop	146
Review and Pursue	150

Creating the HTML Form

As with the previous chapters, the examples in this chapter are based on an HTML form that sends data to a PHP page. In this case, the form is a simple registration page that requests the following information **A**:

- Email address
- Password
- Confirmation of the password
- Year of birth (to verify age)
- Favorite color (for customization purposes)
- Agreement to the site's terms (a common requirement)

The following steps walk through the creation of this form before getting into the PHP code.

To create the HTML form:

1. Begin a new HTML document in your text editor or IDE, to be named **register.php** (Script 6.1):

```
<!DOCTYPE html>  
  
<html lang="en">  
<head>  
  
    <meta charset="utf-8">  
    <title>Registration Form</title>  
  
</head>  
<body>  
    <!-- Script 6.1 - register.html -->  
    <div><p>Please complete this form  
    → to register:</p>
```

Please complete this form to register:

Email Address:

Password:

Confirm Password:

Year You Were Born:

Favorite Color:

☐ I agree to the terms (whatever they may be).

A The HTML form used in this chapter.

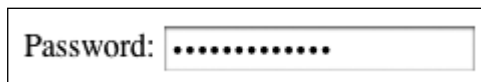
Script 6.1 This pseudo-registration form is the basis for the examples in this chapter.

```
1  <!DOCTYPE html>  
2  <html lang="en">  
3  <head>  
4      <meta charset="utf-8">  
5      <title>Registration Form</title>  
6  </head>  
7  <body>  
8  <!-- Script 6.1 - register.php -->  
9  <div><p>Please complete this form to  
10 <register:</p>  
11  
12 <form action="handle_reg.php"  
    method="post">  
13  
14 <p>Email Address: <input  
    type="email" name="email"  
15 size="30"></p>  
16 <p>Password: <input  
    type="password" name="password"  
17 size="20"></p>  
18 <p>Confirm Password: <input  
    type="password" name="confirm"  
19 size="20" /></p>
```

code continues on next page

Script 6.1 *continued*

```
20    <p>Year You Were Born:
      <input type="date"
        name="year" value="YYYY"
21    size="4"></p>
22    <p>Favorite Color:
23    <select name="color">
24    <option value="">Pick One</option>
25    <option value="red">Red</option>
26    <option value="yellow">Yellow
      </option>
27    <option value="green">Green</option>
28    <option value="blue">Blue</option>
29    </select></p>
30
31    <p><input type="checkbox"
      name="terms" value="Yes"> I agree
      to the terms (whatever they may
      be).</p>
32
33    <input type="submit" name="submit"
      value="Register" />
34
35    </form>
36
37    </div>
38    </body>
39    </html>
```



B A password input type, as it's being filled out.

We are going to create a drop-down menu for the year, when we learn about arrays in Chapter 7. So hold this thought!

2. Create the initial **form** tag:

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

As with many of the previous examples, this page uses the POST method. The handling script, identified by the **action** attribute, will be **handle_reg.php**, found in the same directory as the HTML form.

3. Create inputs for the email address and passwords:

```
<p>Email Address: <input
→ type="email" name="email"
→ size="30"></p>
<p>Password: <input type="password"
name="password" size="20"></p>
<p>Confirm Password: <input
→ type="password" name="confirm" →
size="20"></p>
```

These lines should be self-evident. Each line is wrapped in HTML **<p></p>** tags to improve the spacing in the Web browser. Also, note that two password inputs are created—the second is used to confirm the text entered in the first. Password input types don't reveal what the user enters **B**, so it's a standard policy to require the user to enter passwords twice (thereby ensuring that users know exactly what password they provided).

4. Create an input for the user's birth year:

```
<p>Year You Were Born: <input →
type="date" name="year"
→ value="YYYY" size="4"></p>
```

Rather than use a drop-down menu that displays 50 or 100 years, have users enter their birth year in a text box. By presetting the **value** attribute of the input, you make the text box indicate the proper format for the year **A**.

continues on next page

5. Create a drop-down menu for the user's favorite color:

```
<p>Favorite Color:
<select name="color">
<option value="">Pick One</option>
<option value="red">Red</option>
<option value="yellow">Yellow
→ </option>
<option value="green">Green</option>
<option value="blue">Blue</option>
</select></p>
```

The truth is that I'm adding this input so that it can be used for a specific example later in the chapter, but it might be used to customize the look of the site after the user logs in. Naturally, you can add as many colors as you want here.

6. Create a check box for the user to agree to the site's terms:

```
<p><input type="checkbox"
→ name="terms" value="Yes">
→ I agree to the terms (whatever
→ they may be).</p>
```

Many sites have some sort of terms or licensing that the user must indicate acceptance of, normally by checking a box. This particular form doesn't have a link to where the user can read the terms, but it probably doesn't matter as no one reads them (and this is just

a hypothetical example anyway). In any case, using this element, you'll be able to see how checkboxes are treated by the handling PHP script.

7. Add a submit button and close the form:

```
<input type="submit" name=
→ "submit" value="Register">
</form>
```

8. Complete the HTML page:

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

9. Save the file as **register.php**, place it in the proper directory for your PHP-enabled server, and load the page in your Web browser.

TIP Registration pages should always have users confirm their password and possibly their username or email address (whatever information will be used to log in).

TIP Most registration pages use either a nickname or an email address for the username. If you use the email address as a username, it's easier for your users to remember their registration information (a user may have only a couple of email addresses but a gazillion usernames for different sites around the Web). Furthermore, email addresses are, by their nature, unique to an individual, whereas usernames are not.

The if Conditional

The basic programming conditional is the standard **if** (what used to be called an **if-then** conditional—the **then** is now implied). The syntax for this kind of conditional is simple:

```
if (condition) {  
    statement(s);  
}
```

The *condition* must go within parentheses; then the *statement(s)* are placed within curly brackets. The statements are commands to be executed (for example, printing a string or adding two numbers together). Each separate statement (or command) must have its own semicolon indicating the end of the line, but there is no limit on the number of statements that can be associated with a conditional.

Programmers commonly indent these statements from the initial **if** line to indicate that they're the result of a conditional, but

that format isn't syntactically required. You'll also see people use this syntax:

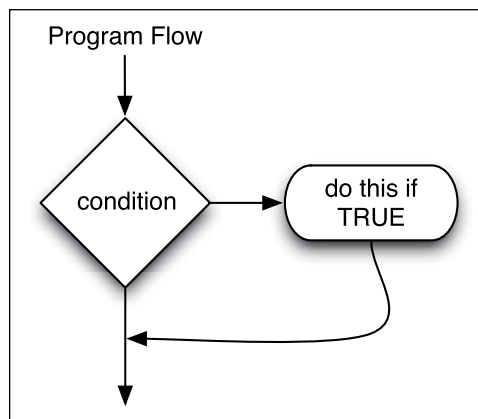
```
if (condition)  
{  
    statement(s);  
}
```

How you arrange your curly brackets is a matter of personal preference—and the source of minor online skirmishes. Just pick a style you like and stick to it.

Failure to use a semicolon after each statement, forgetting an opening or closing parenthesis or curly bracket, or using a semicolon after either of the braces will cause errors to occur. Be mindful of your syntax as you code with conditionals!

PHP uses the Boolean concepts of **TRUE** and **FALSE** when determining whether to execute the statements. If the condition is **TRUE**, the statements are executed; if it's **FALSE**, they are not executed **A**.

continues on next page



A How an **IF** conditional affects the program flow of a script.

Over the course of this chapter (most of it, anyway), a PHP script will be developed until it fully validates the **register.php** form data. To start, this first version of the script will just create the basic shell of the validation process, defining and using a variable with a Boolean value that will track the success of the validation process.

To create an if conditional:

1. Begin a new document in your text editor or IDE, to be named **handle_reg.php** (Script 6.2):

```
<!DOCTYPE html>

<html lang="en">
<head>

    <meta charset="utf-8">
<title>Registration</title>

</head>
<body>
<h1>Registration Results</h1>
```
2. Begin the PHP section and address error management, if necessary:

```
<?php // Script 6.2 - handle_reg.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. Create a *flag* variable:

```
$okay = TRUE;
```

To validate the form data, a *flag* variable will be used to represent whether or not the form was properly

Script 6.2 This shell of a PHP script will be expanded to completely validate the form data.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      http-equiv="Content-Type"
5      <meta
        charset="utf-8">
6      <title>Registration</title>
7  </head>
8  <body>
9  <h1>Registration Results</h1>
10 <?php // Script 6.2 - handle_reg.php
11 /* This script receives seven values
    from register.php
12 email, password, confirm, year, terms,
    color, submit */
13
14 // Address error management, if you
    want.
15
16 // Flag variable to track success:
17 $okay = TRUE;
18
19 // If there were no errors, print a
    success message:
20 if ($okay) {
21     print '<p>You have been
        successfully registered (but
        not really).</p>';
22 }
23 ?>
24 </body>
25 </html>
```

completed. It's called a "flag" variable because the variable stores a simple value that indicates a status. For example: yes, the form was filled out entirely or no, it was not.

The variable is initialized with a Boolean value of `TRUE`, meaning that the assumption is that the form was completed properly. Understand that Booleans are *case-insensitive* in PHP, so you could also write `True` or `true`.

4. Print a message if everything is all right:

```
if ($okay) {  
    print '<p>You have been  
    → successfully registered  
    → (but not really).</p>';  
}
```

Over the course of this chapter, validation routines will be added to this script, checking the submitted form data. If any data fails a routine, then `$okay` will be set to `FALSE`. In that case, this conditional will also be `FALSE`, so the message won't be printed. However, if the data passes every validation routine, then `$okay` will still be `TRUE`, in which case this message will be printed.

Please complete this form to register:

Email Address:

Password:

Confirm Password:

Year You Were Born:

Favorite Color:

☒ I agree to the terms (whatever they may be).

- B** Filling out the HTML form to any degree...

5. Complete the PHP section and the HTML page:

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

6. Save the file as `handle_reg.php`, place it in the proper directory for your PHP-enabled server (in the same directory as `register.php`), and test both in your Web browser **B** and **C**.

Of course, the fact is that this particular script will always print the success message, as no code will set `$okay` to `FALSE`. You can even run the script directly and see the same result.

TIP If the statement area of your conditional is only one line long, you technically don't need the curly brackets. In that case, you can write the conditional using either of these formats:

```
if (condition) statement;
```

or

```
if (condition)  
    statement;
```

You may run across code in these formats. However, I think it's best to always use the multiline format, with the curly brackets (as demonstrated in the syntax introduction) to improve consistency and minimize errors.

Registration Results

You have been successfully registered (but not really).

- C** ...results in just this.

Validation Functions

PHP has dozens of functions commonly used to validate form data. Of these functions, the three most important ones are used in this chapter's examples.

First up is the **empty()** function, which checks to see if a given variable has an "empty" value. A variable is considered to have an empty value if the variable has no value, has a value of 0, or has a value of FALSE. In any of these cases, the function returns TRUE; otherwise, it returns FALSE:

```
$var1 = 0;
$var2 = 'something';
$var3 = ' '; // An empty string
empty($var); // TRUE, no defined value
empty($var1); // TRUE, empty value
empty($var2); // FALSE, non-empty value
empty($var3); // TRUE, empty value
```

This function is perfect for making sure that text boxes in forms have been filled out. For example, if you have a text input named *email* and the user doesn't enter anything in it before submitting the form, then the `$_POST['email']` variable will exist but will have an empty value.

Next is the **isset()** function, which is almost the opposite of **empty()**, albeit with a slight difference. The **isset()** function returns TRUE if a variable has any value (including 0, FALSE, or an empty string). If the variable does not have a value, **isset()** returns FALSE:

```
$var1 = 0;
$var2 = 'something';
$var3 = ' '; // An empty string
isset($var); // FALSE, no defined value
isset($var1); // TRUE
isset($var2); // TRUE
isset($var3); // TRUE
```


Script 6.3 Using `if` conditionals and the `empty()` function, this PHP script checks if email address and password values were provided.

```

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/
   html" xml:lang="en" lang="en">
4  <head>
5    <meta http-equiv="Content-Type"
   content="text/html;
   charset=utf-8"/>
6    <title>Registration</title>
7    <style type="text/css"
   media="screen">
8      .error { color: red; }
9    </style>
10 </head>
11 <body>
12 <h1>Registration Results</h1>
13 <?php // Script 6.3 - handle_reg.php #2
14 /* This script receives seven values
   from register.html:
15 email, password, confirm, year, terms,
   color, submit */
16
17 // Address error management, if you want.
18
19 // Flag variable to track success:
20 $okay = TRUE;
21
22 // Validate the email address:
23 if (empty($_POST['email'])) {
24   print '<p class="error">Please
   enter your email address.</p>';
25   $okay = FALSE;
26 }
27
28 // Validate the password:
29 if (empty($_POST['password'])) {
30   print '<p class="error">Please
   enter your password.</p>';
31   $okay = FALSE;
32 }
33
34 // If there were no errors, print a
   success message:
35 if ($okay) {
36   print '<p>You have been successfully
   registered (but not really).</p>';
37 }
38 ?>
39 </body>
40 </html>

```

The `isset()` function is commonly used to validate nontext form elements like checkboxes, radio buttons, and select menus.

Finally, the `is_numeric()` function returns `TRUE` if the submitted variable has a valid numerical value and `FALSE` otherwise. Integers, decimals, and even strings (if they're a valid number) can all pass the `is_numeric()` test:

```

$var1 = 2309;
$var2 = '80.23';
$var3 = 'Bears';
is_numeric($var1); // TRUE
is_numeric($var2); // TRUE
is_numeric($var3); // FALSE

```

Let's start applying these functions to the PHP script to perform data validation.

To validate form data:

1. Open `handle_reg.php` (Script 6.2) in your text editor or IDE, if it is not already open.
2. Within the document's head, define a CSS class (Script 6.3):

```

<style type="text/css"
→ media="screen">
    .error { color: red; }
</style>

```

This CSS class will be used to format any printed registration errors.

3. Validate the email address:

```

if (empty($_POST['email'])) {
    print '<p class="error">Please
→ enter your email address.</p>';
    $okay = FALSE;
}

```

continues on next page

This **if** conditional uses the code **empty(\$_POST['email'])** as its condition. If that variable is empty, meaning it has no value, a value of 0, or a value of an empty string, the conditional is TRUE. In that case, the **print** statement will be executed and the **\$okay** variable will be assigned a value of FALSE (indicating that everything is not okay).

If the variable isn't empty, then the conditional is FALSE, the **print** function is never called, and **\$okay** will retain its original value.

4. Repeat the validation for the password:

```
if (empty($_POST['password'])) {  
    print '<p class="error">Please  
    → enter your password.</p>';  
    $okay = FALSE;  
}
```

This is a repeat of the email validation, but with the variable name and **print** statement changed accordingly. The other form inputs will be validated in time.

All of the printed error messages are placed within HTML paragraph tags that have a **class** value of **error**. By doing so, the CSS formatting will be applied (i.e., the errors will be printed in red, not that it'll be apparent in this book's figures).

5. Save the file as **handle_reg.php**, place it in the same directory as **register.php** (on your PHP-enabled server), and test both the form and the script in your Web browser **A** and **B**.

Please complete this form to register:

Email Address:

Password:

Confirm Password:

Year You Were Born:

Favorite Color:

☐ I agree to the terms (whatever they may be).

A If you omit the email address or password form input...

Registration Results

Please enter your email address.

Please enter your password.

B ...you'll see messages like these.

6. Resubmit the form in different states of completeness to test the results some more.

If you do provide both email address and password values, the result will be exactly like that in Figure C in the section “The if Conditional,” because the `$okay` variable will still have a value of `TRUE`.

TIP When you use functions within conditionals, as with `empty()` here, it's easy to forget a closing parenthesis and see a parse error. Be extra careful with your syntax when you're coding any control structure.

TIP One use of the `isset()` function is to avoid referring to a variable unless it exists. If PHP is set to report notices (see “Error Reporting” in Chapter 3), then, for example, using `$var` if it has not been defined will cause an error. You can avoid this by coding

```
if (isset($var)) {  
    // Do whatever with $var.  
}
```

TIP Even though almost all form data is sent to a PHP script as strings, the `is_numeric()` function can still be used for values coming from a form because it can handle strings that contain only numbers.

TIP The `isset()` function can take any number of variables as arguments:

```
if (isset($var1, $var2)) {  
    print 'Both variables exist.';  
}
```

If all the named variables are set, the function returns `TRUE`; if any variable is not set, the function returns `FALSE`.

Using else

The next control structure we'll discuss is the **if-else** conditional. This control structure allows you to execute one or more statements when a condition is **TRUE** and execute one or more other statements when the condition is **FALSE**:

```
if (condition) {  
    statement(s);  
} else {  
    other_statement(s);  
}
```

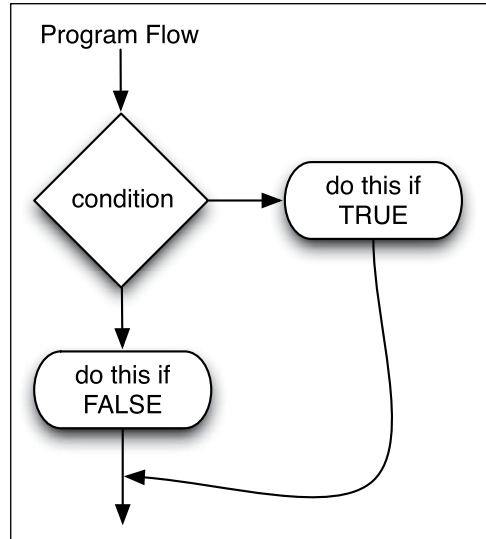
The important thing to remember when using this construct is that unless the condition is explicitly met, the **else** statement will be executed. In other words, the statements after the **else** constitute the *default action*, whereas the statements after the **if** condition are the exception to the rule **A**.

Let's rewrite the **handle_reg.php** page, incorporating an **if-else** conditional to validate the birth year. In the process, a new variable will be created, representing the user's age.

To use else:

1. Open **handle_reg.php** (Script 6.3) in your text editor or IDE, if it is not already open.
2. After the password validation but before the **\$okay** conditional, begin a new conditional (**Script 6.4**):

```
if (is_numeric($_POST['year'])) {  
    Because the year variable should be a  
    number, you can use the is_numeric()  
    function to check its value, rather than  
    empty(). This is a basic start to this  
    particular form element's validation;  
    later scripts will expand on this.
```



A How an **IF-ELSE** conditional affects the program flow of a script.

Script 6.4 By adding an **if-else** conditional, this script validates the birth year and creates a new variable in the process.

```
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>Registration</title>  
7      <style type="text/css"  
        media="screen">  
8          .error { color: red; }  
9      </style>  
10 </head>  
11 <body>  
12 <h1>Registration Results</h1>  
13 <?php // Script 6.4 - handle_reg.php #3  
14 /* This script receives seven values  
    from register.html:  
15 email, password, confirm, year, terms,  
    color, submit */  
16
```

code continues on next page

Script 6.4 *continued*

```
17 // Address error management, if you
18 want.
19 // Flag variable to track success:
20 $okay = TRUE;
21
22 // Validate the email address:
23 if (empty($_POST['email'])) {
24     print '<p class="error">Please
25     enter your email address.</p>';
26     $okay = FALSE;
27 }
28 // Validate the password:
29 if (empty($_POST['password'])) {
30     print '<p class="error">Please
31     enter your password.</p>';
32     $okay = FALSE;
33 }
34 // Validate the birth year:
35 if (is_numeric($_POST['year'])) {
36     $age = date('Y') - $_POST['year'];
37     // Calculate age this year.
38 } else {
39     print '<p class="error">Please
40     enter the year you were born as
41     four digits.</p>';
42     $okay = FALSE;
43 }
44 // If there were no errors, print a
45 success message:
46 if ($okay) {
47     print '<p>You have been successfully
48     registered (but not really).</p>';
49     print "<p>You will turn $age
50     this year.</p>";
51 }
52 ?>
53 </body>
54 </html>
```

$\$age = \text{date}('Y') - \$_POST['year'];$

3. Create a new variable:

\$age = ~~2011~~ - \$_POST['year'];

If the **\$_POST['year']** variable has a numeric value (meaning that the conditional is TRUE), then the **\$age** variable is assigned the value of the current year minus the provided year. For now, without knowledge of PHP's date functions, just hard-code the current year into the equation.

4. Add an **else** clause:

```
} else {
    print '<p class="error">Please
    → enter the year you were born
    → as four digits.</p>';
    $okay = FALSE;
}
```

If the year does not have a numeric value, an error message is printed and the **\$okay** variable is set to FALSE (as is the case if any validation routine fails).

5. After the final **print** statement but within the same **\$okay** conditional, also print out the value of **\$age**:

```
print "<p>You will turn $age this
→ year.</p>";
```

If the **\$okay** variable still has a value of TRUE, then the submitted data passed every validation routine. This means that the user's age has been calculated (in the sense of how old they'll be at some point this year), and it can be printed, too.

continues on next page

6. Save your script, place it in the same directory as **register.php** (on your PHP-enabled server), and test it in your Web browser again **B**, **C**, and **D**.

TIP Another good validation function is `checkdate()`, which you can use to confirm that a date exists (or existed in the past). You would use it like so:

```
if (checkdate($month, $day, $year)) {...
```

Please complete this form to register:

Email Address:

Password:

Confirm Password:

Year You Were Born:

Favorite Color:

☒ I agree to the terms (whatever they may be).

- B** Test the form again, without providing a year value, and...

Registration Results

Please enter the year you were born as four digits.

- C** ...you'll see this.

Registration Results

You have been successfully registered (but not really).

You will turn 57 this year.

- D** If the user provides a numeric value for their birth year, the user's age will now be calculated and printed (assuming that an email address and password was also provided).

TABLE 6.1 PHP's Operators

Operator	Usage	Type
+	Addition	Arithmetic
-	Subtraction	Arithmetic
*	Multiplication	Arithmetic
/	Division	Arithmetic
%	Modulus (remainder of a division)	Arithmetic
++	Incrementation	Arithmetic
--	Decrementation	Arithmetic
=	Assigns a value to a variable	Assignment
==	Equality	Comparison
!=	Inequality	Comparison
<	Less than	Comparison
>	Greater than	Comparison
<=	Less than or equal to	Comparison
>=	Greater than or equal to	Comparison
!	Negation	Logical
AND	And	Logical
&&	And	Logical
OR	Or	Logical
	Or	Logical
XOR	Or not	Logical
.	Concatenation	String

More Operators

Previous chapters discussed most of PHP's operators along with the variable types that use them. These operators include *arithmetic* for numbers: addition (+), subtraction (-), multiplication (*), and division (/), along with the incremental (++) and decremental (--) shortcuts for increasing or decreasing the value of a number by 1. Then there is the *assignment* operator (=), which is used to set the value of a variable, regardless of type. You've also learned about *concatenation* (.), which appends one string to another.

When it comes to creating conditionals, the *comparison* and *logical* operators are the most important. Table 6.1 lists the operators to be discussed, along with those you've already seen.

Comparison

When the assignment operator (the equals sign) was introduced in Chapter 2, "Variables," you learned that its meaning isn't exactly what you'd conventionally think it to be. The line

```
$var = 5;
```

doesn't state that **\$var** is equal to 5 but that it is assigned the value of 5. This is an important distinction.

When you're writing conditionals, you'll often want to see if a variable is equal to a specific value (to match usernames or passwords, perhaps), which you can't do with the equals sign alone (because that operator is used for assigning a value, not equating values). Instead, for comparisons, use the equality operator (==):

```
$var = 5;  
if ($var == 5) { ...
```

continues on next page

These two lines of code together first establish the value of `$var` as 5 and then make a TRUE conditional that checks if `$var` is equal to 5. This example demonstrates the significant difference one more equals sign makes in your PHP code and why *you must distinguish carefully between the assignment and comparison operators.*

The next comparison operator—*not equal to*—is represented by an exclamation mark coupled with an equals sign (`!=`).

The remaining comparison operators are identical to their mathematical counterparts: less than (`<`), greater than (`>`), less than or equal to (`<=`), and greater than or equal to (`>=`).

As a demonstration of comparison operators, you'll check that the user's birth year is before 2011 and that the confirmed password matches the original password.

To use comparison operators:

1. Open `handle_reg.php` (Script 6.4) in your text editor or IDE, if it is not already.
2. After the password validation, check that the two passwords match (Script 6.5):

```
if ($_POST['password']
→ != $_POST['confirm']) {
    print '<p class="error">Your
→ confirmed password does
→ not match the original
→ password.</p>';
    $okay = FALSE;
}
```

To compare these two string values, use the inequality operator. Alternatively, you could use one of the string comparison functions (see Chapter 5, "Using Strings"), but `!=` is just fine.

Script 6.5 This version of the form-handling script uses comparison operators to validate the password and year values.

```
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>Registration</title>
7 <style type="text/css" media="screen">
8 .error { color: red; }
9 </style>
10 </head>
11 <body>
12 <h1>Registration Results</h1>
13 <?php // Script 6.5 - handle_reg.php #4
14 /* This script receives seven values
from register.html:
15 email, password, confirm, year, terms,
color, submit */
16
17 // Address error management, if you want.
18
19 // Flag variable to track success:
20 $okay = TRUE;
21
22 // Validate the email address:
23 if (empty($_POST['email'])) {
24     print '<p class="error">Please enter
your email address.</p>';
25     $okay = FALSE;
26 }
27
28 // Validate the password:
29 if (empty($_POST['password'])) {
30     print '<p class="error">Please enter
your password.</p>';
31     $okay = FALSE;
32 }
33
34 // Check the two passwords for equality:
35 if ($_POST['password'] !=
$_POST['confirm']) {
36     print '<p class="error">Your
confirmed password does not match
the original password.</p>';
```

code continues on next page

Script 6.5 *continued*

```
37     $okay = FALSE;
38 }
39
40 // Validate the birth year:
41 if (is_numeric($_POST['year'])) {
42     $age = 2011 - $_POST['year'];
43     // Calculate age this year.
44 } else {
45     print '<p class="error">Please enter
46         the year you were born as four
47         digits.</p>';
48     $okay = FALSE;
49 }
50 // Check that they were born before
51 // this year:
52 if ($_POST['year'] >= 2011) {
53     print '<p class="error">Either you
54         entered your birth year wrong or
55         you come from the future!</p>';
56     $okay = FALSE;
57 }
58
59 // If there were no errors, print a
60 // success message:
61 if ($okay) {
62     print '<p>You have been successfully
63         registered (but not really).</p>';
64     print "<p>You will turn $age this
65         year.</p>";
66 }
67
68 ?>
69 </body>
70 </html>
```

3. After the year validation, report an error if the year is greater than or equal to 2011:

```
if ($_POST['year'] >= date('Y'))2011) {
    print '<p class="error">Either
        → you entered your birth year
        → wrong or you come from the
        → future!</p>';
    $okay = FALSE;
}
```

If the user entered their year of birth as 2011 or later, it's presumably a mistake. (If you're reading this book after 2011, change the year accordingly).

continues on next page

4. Save your script, place it in the same directory as **register.php** (on your PHP-enabled server), and test it in your Web browser again **A** and **B**.

TIP Before you compare two string values that come from a form (like the password and confirmed password), it's a good idea to apply the `trim()` function to both, to get rid of any errant spaces. I didn't do so here, so as not to overcomplicate matters, but this habit is recommended.

TIP Another method of checking that a text input type has been filled out (as opposed to using the `empty()` function) is this:

```
if (strlen($var) > 0 ) {  
    // $var is okay.  
}
```

TIP In an `if` conditional, if you make the mistake of writing `$var = 5` in place of `$var == 5`, you'll see that the corresponding conditional statements are always executed. This happens because although the condition `$var == 5` may or may not be `TRUE`, the condition `$var = 5` is always `TRUE`.

TIP Some programmers advocate *reverse conditionals*—for example, writing

```
if (5 == $var) {
```

Although it looks awkward, if you inadvertently code `5 = $var`, an error results (allowing you to catch the mistake more easily) because the number 5 can't be assigned another value.

Please complete this form to register:

Email Address:

Password:

Confirm Password:

Year You Were Born:

Favorite Color:

☒ I agree to the terms (whatever they may be).

- A** Run the form once again...

Registration Results

Your confirmed password does not match the original password.
Either you entered your birth year wrong or you come from the future!

- B** ...with two new validation checks in place.

Nesting Conditionals

Besides using logical operators to create more complex conditionals, you can use *nesting* for this purpose (the process of placing one control structure inside another). The key to doing so is to place the interior conditional as the *statement(s)* section of the exterior conditional. For example:

```
if (condition1) {
    if (condition2) {
        statement(s)2;
    } else { // condition2
else
    other_statement(s)2;
    } // End of 2
} else { // condition1 else
    other_statement(s)1;
} // End of 1
```

As you can see from this example, you can cut down on the complexity of these structures by using extensive indentations and comments. As long as every conditional is syntactically correct, there are no rules as to how many levels of nesting you can have, whether you use an **else** clause or even whether a sub-conditional is part of the **if** or the **else** section of the main conditional.

Logical

Writing conditions in PHP comes down to identifying TRUE or FALSE situations. You can do this by using functions and comparative operators, as you've already seen. *Logical operators—the final operator type discussed in this chapter—help you create more elaborate or obvious constructs.*

In PHP, one example of a TRUE condition is simply a variable name that has a value that isn't zero, an empty string, or FALSE, such as

```
$var = 5;
if ($var) { ...
```

You've already seen this with the **\$okay** variable being used in the handling PHP script.

A condition is also TRUE if it makes logical sense:

```
if (5 >= 3) { ...
```

A condition will be FALSE if it refers to a variable and that variable has no value (or a value of 0 or an empty string), or if you've created an illogical construct. The following condition is always FALSE:

```
if (5 <= 3) { ...
```

*In PHP, the exclamation mark (!) is the **not** operator. You can use it to invert the TRUE/FALSE status of a statement. For example:*

```
$var = 'value';
if ($var) {... // TRUE
if (!$var) {... // FALSE
if (isset($var)) {... // TRUE
if (!isset($var)) {... // FALSE
if (!empty($var)) {... // TRUE
```

To go beyond simple one-part conditions, PHP supports five more types of logical operators: two versions of *and* (**AND** and **&&**), two versions of *or* (**OR** and **||**—a character called the *pipe*, put together twice), and

continues on next page

Exclusive OR: If both conditions are true, the operation will not execute.



or not (XOR). When you have two options for one operator (as with *and* and *or*), they differ only in precedence. For almost every situation, you can use either version of *and* or either version of *or* interchangeably.

Using parentheses and logical operators, you can create even more complex **if** conditionals. For an **AND** conditional, every conjoined part must be TRUE in order for the whole conditional to be TRUE. With **OR**, at least one subsection must be TRUE to render the whole condition TRUE. These conditionals are TRUE:

```
if ( (5 <= 3) OR (5 >= 3) ) { ...  
if ( (5 > 3) AND (5 < 10) ) { ...
```

These conditionals are FALSE:

```
if ( (5 != 5) AND (5 > 3) ) { ...  
if ( (5 != 5) OR (5 < 3) ) { ...
```

As you construct your conditionals, remember two important things: first, in order for the statements that are the result of a conditional to be executed, the entire conditional must have a TRUE value; second, by using parentheses, you can ignore rules of precedence and ensure that your operators are addressed in the order of your choosing.

To demonstrate logical operators, let's add more conditionals to the **handle_reg.php** page. You'll also nest one of the year conditionals inside another conditional (see the sidebar "Nesting Conditionals" for more).

To use logical operators:

1. Open **handle_reg.php** (Script 6.5) in your text editor or IDE, if it is not already open.
2. Delete the existing year validations (Script 6.6).

You'll entirely rewrite these conditionals as one nested conditional, so it's best to get rid of the old versions entirely.

Script 6.6 Here the handling PHP script is changed so that the year validation routine uses both multiple and nested conditions. Also, the terms of agreement check box is now validated.

```
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>Registration</title>  
7      <style type="text/css" media="screen">  
8          .error { color: red; }  
9      </style>  
10 </head>  
11 <body>  
12 <h1>Registration Results</h1>  
13 <?php // Script 6.6 - handle_reg.php  
14 /* This script receives seven values  
from register.html:  
15 email, password, confirm, year, terms,  
color, submit */  
  
16  
17 // Address error management, if you want.  
18  
19 // Flag variable to track success:  
20 $okay = TRUE;  
21  
22 // Validate the email address:  
23 if (empty($_POST['email'])) {  
24     print '<p class="error">Please enter  
your email address.</p>';  
25     $okay = FALSE;  
26 }  
27  
28 // Validate the password:  
29 if (empty($_POST['password'])) {  
30     print '<p class="error">Please enter  
your password.</p>';  
31     $okay = FALSE;  
32 }  
33  
34 // Check the two passwords for equality:  
35 if ($_POST['password'] !=  
$_POST['confirm']) {  
36     print '<p class="error">Your confirmed  
password does not match the original  
password.</p>';
```

code continues on next page

Script 6.6 *continued*

```
37     $okay = FALSE;
38 }
39
40 // Validate the year:
41 if ( is_numeric($_POST['year']) AND
    (strlen($_POST['year']) == 4) ) {
42
43     // Check that they were born
    before 2011.
44     if ($_POST['year'] < date('Y')) {
45         $age = date('Y') - $_POST['year'];
46         // Calculate age this year.
47     } else {
48         print '<p class="error">Either
        you entered your birth year
        wrong or you come from the
        future!</p>';
49         $okay = FALSE;
50     } // End of 2nd conditional.
51 } else { // Else for 1st conditional.
52
53     print '<p class="error">Please
        enter the year you were born as
        four digits.</p>';
54     $okay = FALSE;
55 } // End of 1st conditional.
56
57 // Validate the terms:
58 if (!isset($_POST['terms'])) {
59     print '<p class="error">You must
        accept the terms.</p>';
60     $okay = FALSE;
61 }
62
63 // If there were no errors, print a
    success message:
64 if ($okay) {
65     print '<p>You have been successfully
        registered (but not really).</p>';
66     print "<p>You will turn $age this
        year.</p>";
67 }
68 }
69 >>
70 </body>
71 </html>
```

3. Check that the year variable is a four-digit number:

```
if ( is_numeric($_POST['year']) AND
    → (strlen($_POST['year']) == 4) ) {
```

This conditional has two parts. The first you've already seen—it tests for a valid numeric value. The second part gets the length of the year variable (using the `strlen()` function) and checks if the length value is equal to 4. Because of the **AND**, this conditional is **TRUE** only if both conditions are met.

4. Create a subconditional to check if the year value is before 2011:

```
if ($_POST['year'] < date('Y')) {
    $age = date('Y') - $_POST['year'];
} else {
    print '<p class="error">Either
    → you entered your birth year
    → wrong or you come from the
    → future!</p>';
    $okay = FALSE;
} // End of 2nd conditional.
```

This **if-else** conditional acts as the *statements* part of the main conditional, and is thus executed only if that condition is **TRUE**. This **if-else** checks whether the year variable is less than 2011 (i.e., the user must have been born before the current year). If that condition is **TRUE**, the user's age is calculated as before. Otherwise, an error message is printed and the **\$okay** variable is set to **FALSE** (indicating a problem occurred).

Note that this conditional is just the opposite of the previous version: verifying that a value is less than some number instead of greater than or equal to that number.

continues on next page

5. Complete the main year conditional:

```
} else { // Else for 1st
→ conditional.
    print '<p class="error">Please
    → enter the year you were born
    → as four digits.</p>';
    $okay = FALSE;
} // End of 1st conditional.
```

This **else** section completes the conditional begun in Step 3. If at least one of the conditions set forth there is **FALSE**, this message is printed and **\$okay** is set to **FALSE**.

6. Confirm that the terms checkbox wasn't ignored:

```
if (!isset($_POST['terms'])) {
    print '<p class="error">You must
    → accept the terms.</p>';
    $okay = FALSE;
}
```

If the **\$_POST['terms']** variable is not set, then the user failed to check that box and an error should be reported. To be more exact, this conditional could be

```
if ( !isset($_POST['terms']) OR
→ ($_POST['terms'] != 'Yes') ) {
```

7. Those are the only changes to the script, so you can now save it again, place it in the same directory as **register.php** (on your PHP-enabled server), and test it in your Web browser again **C** and **D**.

Please complete this form to register:

Email Address:

Password:

Confirm Password:

Year You Were Born:

Favorite Color:

☐ I agree to the terms (whatever they may be).

- C** The PHP script now catches if the year isn't a four-digit number, as will be the case with this form submission.

Registration Results

Please enter the year you were born as four digits.

You must accept the terms.

- D** Error messages are printed if fields are incorrectly filled out or if the terms checkbox is not checked.

Book Errata corrected!



8. If desired, change your **year** value to be in the future and submit the form again **E**.

TIP It's another common programming convention—which is maintained in this book—to write the terms **TRUE** and **FALSE** in all capitals. This isn't a requirement of PHP, though. For example, the following conditional is **TRUE**:

```
if (true) {...
```

TIP It's very easy in long, complicated conditionals to forget an opening or closing parenthesis or curly bracket, which will produce either error messages or unexpected results. Find a system (like spacing out your conditionals and using comments) to help clarify your code. Another good technique is to create the entire conditional's structure first, and then go back to add the details.

`var_dump()` or `print_r()` will help you to problem solve your code

TIP If you have problems getting your **if-else** statements to execute, print out the values of your variables to help debug the problem. A conditional may not be **TRUE** or **FALSE** because a variable doesn't have the value you think it does.

Registration Results

Either you entered your birth year wrong or you come from the future!

- E** The year validation still checks that the date is before 2011.

Using elseif

Similar to the **if-else** conditional is **if-elseif** (or **if-elseif-else**). This conditional acts like a running **if** statement and can be expanded to whatever complexity you require:

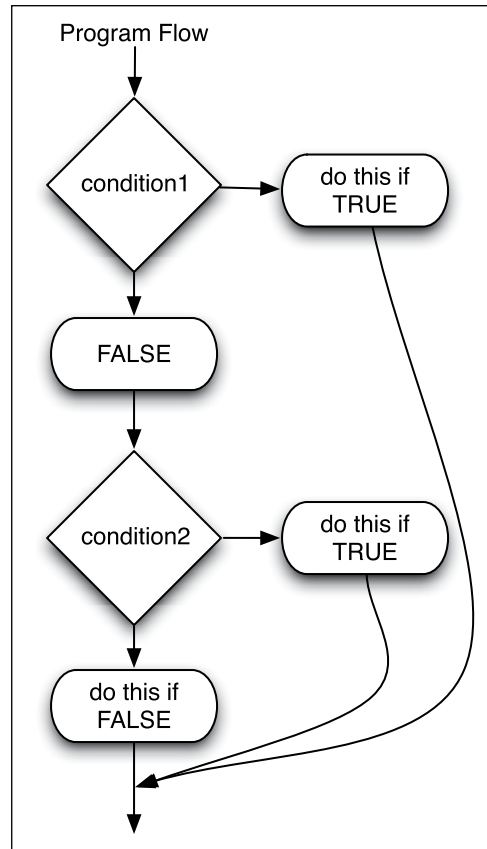
```
if (condition1) {  
    statement(s);  
} elseif (condition2) {  
    other_statement(s);  
}
```

Here's another example **A**:

```
if (condition1) {  
    statement(s);  
} elseif (condition2) {  
    other_statement(s);  
} else {  
    other_other_statement(s);  
}
```

If present, you must always make the **else** the last part of a conditional because it's executed unless one of the conditions to that point has been met (again, **else** represents the *default* behavior). You can, however, continue to use **elseif**s as many times as you want as part of one **if** conditional. You may also forego an **else** clause, if you don't need a default response.

As an example of this, let's create a conditional that prints a message based on the selected color value.



A How an **IF-ELSEIF-ELSE** conditional affects the program flow of a script.

Script 6.7 This multiline **if-elseif-else** conditional validates that a submitted color has an allowed value and is used to determine what type of color the selection is.

```

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>Registration</title>
7    <style type="text/css" media="screen">
8      .error { color: red; }
9    </style>
10 </head>
11 <body>
12 <h1>Registration Results</h1>
13 <?php // Script 6.7 - handle_reg.php #6
14 /* This script receives seven values
from register.html:
15 email, password, confirm, year, terms,
color, submit */
16
17 // Address error management, if you want.
18
19 // Flag variable to track success:
20 $okay = TRUE;
21
22 // Validate the email address:
23 if (empty($_POST['email'])) {
24     print '<p class="error">Please enter
your email address.</p>';
25     $okay = FALSE;
26 }
27
28 // Validate the password:
29 if (empty($_POST['password'])) {
30     print '<p class="error">Please enter
your password.</p>';
31     $okay = FALSE;
32 }
33
34 // Check the two passwords for equality:
35 if ($_POST['password'] !=
$_POST['confirm']) {
36     print '<p class="error">Your confirmed
password does not match the original
password.</p>';

```

code continues on next page

To use elseif:

1. Open **handle_reg.php** (Script 6.6) in your text editor or IDE, if it is not already open.
2. Before the **\$okay** conditional, begin a new conditional (**Script 6.7**):

```

if ($_POST['color'] == 'red') {
    $color_type = 'primary';

```

The color value comes from a select menu with four possible options: *red*, *yellow*, *green*, and *blue*. This conditional will determine whether the user has selected a primary—red, yellow, or blue—or secondary (all others) color. The first condition checks if the value of **\$_POST['color']** is equal to the string *red*.

Be certain to use the *equality* operator—two equals signs—and not the *assignment* operator—one—in the conditional.

3. Add an **elseif** clause for the second color:

```

} elseif ($_POST['color'] ==
→ 'yellow') {
    $color_type = 'primary';

```

The **elseif** continues the main conditional begun in Step 2. The condition itself is a replication of the condition in Step 2, using a new color comparison.

continues on next page

4. Add **elseif** clauses for the other two colors:

```
} elseif ($_POST['color'] ==
→ 'green') {
    $color_type = 'secondary';
} elseif ($_POST['color'] ==
→ 'blue') {
    $color_type = 'primary';
```

Once you understand the main concept, it's just a matter of repeating the **elseif**s for every possible color value.

5. Add an **else** clause:

```
} else {
    print '<p class="error">Please
→ select your favorite
→ color.</p>';
    $okay = FALSE;
}
```

If the user didn't select a color, or if they manipulated the form to submit a different color value (other than *red*, *yellow*, *green*, or *blue*), none of the conditions will be TRUE, meaning this **else** clause will take effect. That clause prints an error and assigns a value of FALSE to **\$okay**, indicating a problem.

It doesn't matter in what order the colors are checked, so long as the **else** clause comes last.

Script 6.7 *continued*

```
37     $okay = FALSE;
38 }
39
40 // Validate the year:
41 if ( is_numeric($_POST['year']) AND
    (strlen($_POST['year']) == 4) ) {
42
43     // Check that they were born before
    2011.
44     if ($_POST['year'] < 2011) {
45         $age = 2011 - $_POST['year'];
46         // Calculate age this year.
47     } else {
48         print '<p class="error">Either you
    entered your birth year wrong or
    you come from the future!</p>';
49         $okay = FALSE;
50     } // End of 2nd conditional.
51 } else { // Else for 1st conditional.
52
53     print '<p class="error">Please enter
    the year you were born as four
    digits.</p>';
54     $okay = FALSE;
55
56 } // End of 1st conditional.
57
58 // Validate the terms:
59 if ( !isset($_POST['terms']) AND
    ($_POST['terms'] == 'Yes') ) {
60     print '<p class="error">You must
    accept the terms.</p>';
61     $okay = FALSE;
62 }
63
64 // Validate the color:
65 if ($_POST['color'] == 'red') {
66     $color_type = 'primary';
67 } elseif ($_POST['color'] == 'yellow') {
68     $color_type = 'primary';
69 } elseif ($_POST['color'] == 'green') {
70     $color_type = 'secondary';
71 } elseif ($_POST['color'] == 'blue') {
72     $color_type = 'primary';
73 } else { // Problem!
74     print '<p class="error">Please
    select your favorite color.</p>';
75     $okay = FALSE;
76 }
```

code continues on next page

Script 6.7 *continued*

```
77
78 // If there were no errors, print a
    success message:
79 if ($okay) {
80     print '<p>You have been successfully
        registered (but not really).</p>';
81     print "<p>You will turn $age this
        year.</p>";
82     print "<p>Your favorite color is a
        $color_type color.</p>";
83 }
84 ?>
85 </body>
86 </html>
```

Registration Results

You have been successfully registered (but not really).

You will turn 12 this year.

Your favorite color is a secondary color.

B The script now prints a message acknowledging the user's color choice.

Registration Results

Please select your favorite color.

C Failure to select a color results in this error message.

6. Within the **\$okay** conditional, print the user's favorite color type:

```
print "<p>Your favorite color is
→ a $color_type color.</p>";
```

7. Save the script, place it in the same directory as **register.html** (on your PHP-enabled server), and test it in your Web browser again, using different color options **B** and **C**.

TIP One thing most beginner developers don't realize is that it's possible—in fact, quite easy—for a hacker to submit data to your PHP script without using your intended HTML form. For this reason, it's important that you validate the existence of expected variables (i.e., that they are set), their type, and their values.

TIP PHP also allows you to write **elseif** as two words, if you prefer:

```
if (condition1) {
    statement(s);
} else if (condition2) {
    statement(s)2;
}
```

Alternative to the If, Elseif, Else statement. There is no particular advantage to use one statement over the other.

The Switch Conditional

Once you get to the point where you have longer **if-elseif-else** conditionals, you may find that you can save programming time and clarify your code by using a **switch** conditional instead. The **switch** conditional takes only one possible condition, normally just a variable:

```
switch ($var) {
    case value1:
        statement(s)1;
        break;
    case value2:
        statement(s)2;
        break;
    default:
        statement(s)3;
        break;
}
```

You must understand how a **switch** conditional works in order to use it properly. After the keyword **switch**, a variable is identified within parentheses. PHP will then look at each case in order, trying to identify a matching value. Note that, as with any other use of strings and numbers in PHP, numeric values would not be quoted; string values should be. After the **case value** section, a *colon* (not a semicolon) prefaces the associated statements, which are normally indented beginning on the following line.

Once PHP finds a case that matches the value of the conditional variable, it executes the subsequent statements. Here's the tricky part: once PHP has found a matching case, it will continue going through the **switch** until it either comes to the end of the **switch** conditional (the closing curly bracket) or hits a **break** statement, at which point it exits the **switch** construct. Thus, it's imperative that you

Script 6.8 Switch conditionals can simplify complicated **if-elseif** conditionals.

```
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>Registration</title>
7      <style type="text/css" media="screen">
8          .error { color: red; }
9      </style>
10 </head>
11 <body>
12 <h1>Registration Results</h1>
13 <?php // Script 6.8 - handle_reg.php #7
14 /* This script receives seven values
        from register.html:
15 email, password, confirm, year, terms,
        color, submit */
16
17 // Address error management, if you want.
18
19 // Flag variable to track success:
20 $okay = TRUE;
21
22 // Validate the email address:
23 if (empty($_POST['email'])) {
24     print '<p class="error">Please enter
        your email address.</p>';
25     $okay = FALSE;
26 }
27
28 // Validate the password:
29 if (empty($_POST['password'])) {
30     print '<p class="error">Please enter
        your password.</p>';
31     $okay = FALSE;
32 }
33
34 // Check the two passwords for equality:
35 if ($_POST['password'] !=
    $_POST['confirm']) {
36     print '<p class="error">Your confirmed
        password does not match the original
        password.</p>';
37     $okay = FALSE;
38 }
```

code continues on next page

Script 6.8 *continued*

```
39
40 // Validate the year:
41 if ( is_numeric($_POST['year']) AND
    (strlen($_POST['year']) == 4) ) {
42
43     // Check that they were born before
    2011.
44     if ($_POST['year'] < 2011) {
45         $age = 2011 - $_POST['year'];
46         // Calculate age this year.
47     } else {
48         print '<p class="error">Either you
        entered your birth year wrong or
        you come from the future!</p>';
49         $okay = FALSE;
50     } // End of 2nd conditional.
51 } else { // Else for 1st conditional.
52
53     print '<p class="error">Please enter
        the year you were born as four
        digits.</p>';
54     $okay = FALSE;
55 } // End of 1st conditional.
56
57 // Validate the terms:
58 if ( !isset($_POST['terms']) AND
    ($_POST['terms'] == 'Yes') ) {
59     print '<p class="error">You must
        accept the terms.</p>';
60     $okay = FALSE;
61 }
62
63 // Validate the color:
64 switch ($_POST['color']) {
65     case 'red':
66         $color_type = 'primary';
67         break;
68     case 'yellow':
69         $color_type = 'primary';
70         break;
71     case 'green':
72         $color_type = 'secondary';
73         break;
74     case 'blue':
75         $color_type = 'primary';
76         break;
77     default:
```

code continues on next page

close every case—even the **default** case, for consistency’s sake—with a **break** (the sidebar “Break, Exit, Die, and Continue” discusses this keyword in more detail).

This previous **switch** conditional is like a rewrite of:

```
if ($var == value1) {
    statement(s)1;
} elseif ($variable == value2) {
    statement(s)2;
} else {
    statement(s)3;
}
```

Because the **switch** conditional uses the value of **\$var** as its condition, it first checks to see if **\$var** is equal to *value1* and, if so, executes *statement(s)1*. If not, it checks to see if **\$var** is equal to *value2* and, if so, executes *statement(s)2*. If neither condition is met, the default action of the **switch** conditional is to execute *statement(s)3*.

With this in mind, let’s rewrite the colors conditional as a **switch**.

To use a switch conditional:

1. Open **handle_reg.php** (Script 6.7) in your text editor or IDE, if it is not already open.
2. Delete the extended colors conditional (Script 6.8).
3. Begin the **switch**:

```
switch ($_POST['color']) {
```

As mentioned earlier, a **switch** conditional takes only one condition: a variable’s name. In this example, it’s **\$_POST['color']**.

continues on next page

4. Create the first case:

```
case 'red':  
    $color_type = 'primary';  
    break;
```

The first case checks to see if `$_POST['color']` has a value of *red*. If so, then, the same statement is executed as before. Next you include a **break** statement to exit the **switch**.

5. Add a case for the second color:

```
case 'yellow':  
    $color_type = 'primary';  
    break;
```

6. Add cases for the remaining colors:

```
case 'green':  
    $color_type = 'secondary';  
    break;  
case 'blue':  
    $color_type = 'primary';  
    break;
```

Script 6.8 *continued*

```
79     print '<p class="error">Please  
    select your favorite color.</p>';  
80     $okay = FALSE;  
81     break;  
82 } // End of switch.  
83  
84 // If there were no errors, print a  
    success message:  
85 if ($okay) {  
86     print '<p>You have been successfully  
        registered (but not really).</p>';  
87     print "<p>You will turn $age this  
        year.</p>";  
88     print "<p>Your favorite color is a  
        $color_type color.</p>";  
89 }  
90 ?>  
91 </body>  
92 </html>
```

The sidebar erroneously says that **break** can be used to exit an if-else conditional. **Break** only applies to loops and **switch**.

Break, Exit, Die, and Continue

PHP includes many *language constructs*—tools that aren't functions but still do something in your scripts. For example, **print** is a language construct. Another example is **break**, which is demonstrated in the **switch**. **break** exits the current structure, be it a **switch**, an if-else conditional, or a loop.

Similar to this is **continue**, which terminates the current iteration of a loop. Any remaining statements within the loop aren't executed, but the loop's condition is checked again to see if the loop should be entered.

exit and **die** are more potent versions of **break** (and they're synonymous). Instead of exiting the current structure, these two language constructs terminate the execution of the PHP script. Therefore, all PHP code after a use of **exit** or **die** is never executed. For that matter, any HTML after these constructs is never sent to the Web browser. You'll see **die** used most frequently as a heavy-handed error handling tool. **exit** is often used in conjunction with the **header()** function.

Registration Results

You have been successfully registered (but not really).

You will turn 47 this year.

Your favorite color is a primary color.

A The handling script still works the same, whether the user selects a color...

Registration Results

Please select your favorite color.

B ...or fails to.

7. Add a **default** case and complete the switch:

```
default:
    print '<p class="error">Please
    → select your favorite
    → color.</p>';
    $okay = FALSE;
    break;
} // End of switch.
```

This **default** case is the equivalent of the **else** clause used in the original conditional.

8. Save your script, place it in the same directory as **register.html** (on your PHP-enabled server), and test it in your Web browser again **A** and **B**.

TIP A default case isn't required in your switch conditional (you could set it up so that if the value isn't explicitly met by one of the cases, nothing happens), but if it's used, it should be the last case.

TIP If you're using a string in your switch conditional as the case value, keep in mind that it's case sensitive, meaning that *Value* won't match *value*.

The for Loop

Loops are the final type of control structure discussed in this chapter. As suggested earlier, loops are used to execute a section of code repeatedly. You may want to print something a certain number of times, or you may want to do something with each value in an array (i.e., a list of values). For either of these cases, and many more, you can use a loop. (The latter example is demonstrated in the next chapter.)

PHP supports three kinds of loops: **for**, **while**, and **foreach**. The **while** loop is similar to **for**, but it's used most frequently when retrieving values from a database or reading from a text file (it's introduced in the sidebar). The **foreach** loop is related to using arrays and is introduced in the next chapter.

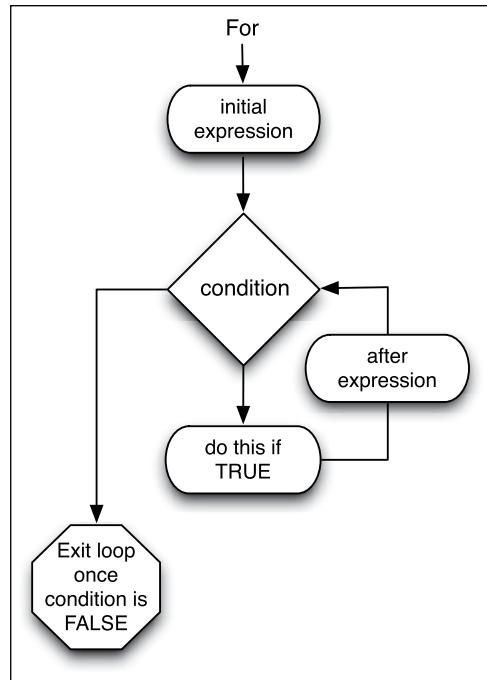
The **for** loop is designed to perform specific statements for a determined number of iterations (unlike **while**, which runs until a condition is FALSE—similar, but significantly different, concepts). You normally use a dummy variable in the loop for this purpose:

```
for (initial expression; condition;  
→ closing expression) {  
    statement(s);  
}
```

The initial expression is executed once: the first time the loop is called. Then the condition is used to determine whether to execute the statements. The closing expression is executed each time the condition is found to be TRUE, but only after the statements are executed **A**.

Here's a simple loop that prints out the numbers 1 through 10:

```
for ($i = 1; $i <= 10; $i++) {  
    print $i;  
}
```



A This flowchart represents how a **for** loop is executed in PHP.

By convention, `$i`, `$j`, and `$k` are frequently used to keep count of the number of times a loop as run. See Tip on page 149

```
for (initialize; test; increment) {  
    statement;  
}
```


Script 6.9 This script uses a PHP **for** loop to dynamically generate the day of the month drop-down menu.

```
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; charset=utf-8"/>
7   <title>Registration Form</title>
8 </head>
9 <!-- Script 6.9 - register.php -->
10 <div><p>Please complete this form to
  register:</p>
11
12 <form action="handle_reg.php"
  method="post">
13
14   <p>Email Address: <input type="text"
15     name="email" size="30" /></p>
16
17   <p>Password: <input type="password"
18     name="password" size="20" /></p>
19
20   <p>Date Of Birth:
21   <select name="month">
22     <option value="">Month</option>
23     <option value="1">January</option>
24     <option value="2">February</option>
25     <option value="3">March</option>
26     <option value="4">April</option>
27     <option value="5">May</option>
28     <option value="6">June</option>
29     <option value="7">July</option>
30     <option value="8">August</option>
31     <option value="9">September</option>
32     <option value="10">October</option>
33     <option value="11">November</option>
34     <option value="12">December</option>
35   </select>
36   <select name="day">
37     <option value="">Day</option>
38   <?php // Print out 31 days:
```

code continues on next page

To practice with the **for** loop, let's expand the registration form so that it asks the user for their complete birthday. A **for** loop can be used to easily create a day drop-down menu in the HTML form.

To write a for loop:

1. Open **register.html** (Script 6.1) in your text editor or IDE, if it is not already open.
2. Remove the existing birth year prompt and input (**Script 6.9**).

You'll replace this one prompt with three separate elements to represent the entire birthday: month, day, and year.
3. Where the birth year prompt was, after the password confirmation and before the color option, add a prompt and a list of months:

```
<p>Date Of Birth:
<select name="month">
<option value="">Month</option>
<option value="1">January</option>
<option value="2">February</option>
<option value="3">March</option>
<option value="4">April</option>
<option value="5">May</option>
<option value="6">June</option>
<option value="7">July</option>
<option value="8">August</option>
<option value="9">September</option>
<option value="10">October</option>
<option value="11">November</option>
<option value="12">December</option>
</select>
```

First is a textual prompt, telling the user to supply their entire date of birth. Next comes a select menu in which the user can pick their birth month. The value for each option is numeric; the viewed text is a string (the month's name).

continues on next page

4. Begin a select menu for the birth day:

```
<select name="day">
<option value="">Day</option>
```

This code starts the **select** form element for the user's birth day of the month. The list of possible values will be generated using PHP.

5. Create a new PHP section:

```
<?php
```

Because PHP can be embedded within HTML, you'll use it to populate the drop-down menu. Begin with the standard PHP tag.

6. Create a **for** loop to print out 31 days as select menu options:

```
for ($i = 1; $i <= 31; $i++) {
    print "<option value=\"$i\">$i
    → </option>\n";
}
```

The loop begins by creating a dummy variable called **\$i**. On the first use of the loop, this variable is set to 1. Then, as long as **\$i** is less than or equal to 31, the contents of the loop are executed. These contents are the **print** line, which creates code like

```
<option value="1">1</option>
```

followed by a return (created with **\n**). After this statement is executed, the **\$i** variable is incremented by 1. Then the condition (**\$i <= 31**) is checked again, and the process is repeated.

7. Close the PHP section:

```
?>
```

8. Save the file as **register.php**.

You must save the file with the **.php** extension now, in order for the PHP code to be executed.

Script 6.9 *continued*

```
39     for ($i = 1; $i <= 31; $i++) {
40         print "<option value=\"$i\">$i
           </option>\n";
41     }
42     ?>
43 </select>
44 <input type="text" name="year"
           value="YYYY" size="4" /></p>
45
46 <p>Favorite Color:
47 <select name="color">
48 <option value="">Pick One</option>
49 <option value="red">Red</option>
50 <option value="yellow">Yellow</option>
51 <option value="green">Green</option>
52 <option value="blue">Blue</option>
53 </select></p>
54
55 <p><input type="checkbox"
           name="terms" value="Yes" /> I agree to
           the terms (whatever they may be).</p>
56
57 <input type="submit" name="submit"
           value="Register" />
58
59 </form>
60
61 </div>
62 </body>
63 </html>
```

Please complete this form to register:

Email Address:

Password:

Confirm Password:

Date Of Birth:

Favorite Color:

☐ I agree to the terms (whatever they may be).

B The new version of the HTML form, with some dynamically generated content.

```

        <option value="11">November</option>
        <option value="12">December</option>
    </select>
    <select name="day">
        <option value="">Day</option>
        <option value="1">1</option>
    <option value="2">2</option>
    <option value="3">3</option>
    <option value="4">4</option>
    <option value="5">5</option>
    <option value="6">6</option>
    <option value="7">7</option>

```

C If you view the HTML source code for the form, you'll see the data generated by the **for** loop.

The while Loop

The second of the three types of loops that exist in PHP—the **while** loop—is designed to continue working as long as the condition you establish is TRUE. Like the **for** loop, it checks the value of the condition before each iteration. Once the condition becomes FALSE, the **while** loop is exited:

```

while (condition) {
    statement(s);
}

```

The main difference between **for** and **while** is that **while** doesn't include a system for setting initial conditions or for executing closing expressions.

You also have the option of using the **do...while** loop, which guarantees that the statements are executed at least once (this isn't necessarily true of the **while** loop):

```

do {
    statement(s);
} while (condition);

```

Although there is a fair amount of overlap regarding when you can use the two major loop constructs (**while** and **for**), you'll discover as you program that sometimes one is more logical than the other. The **while** loop is frequently used in the retrieval of data from a database (see Chapter 12, "Intro to Databases").

- Place the file in the proper directory for your PHP-enabled server and test it in your Web browser **B**.

As long as this script is in the same directory as **handle_reg.php**, you can even fill out and submit the form as you would with the plain HTML version.

- If desired, view the HTML source code to see the PHP-generated options **C**.

TIP It's conventional to use simple variables as the counters within **for** loops: **\$i**, **\$j**, **\$k**, etc.

TIP Just as you can write the **if** conditional on one line if you have only one statement, you can do the same with the **while** and **for** loops. Again, though, this isn't recommended.

TIP Loops can be nested inside each other. You can also place conditionals within loops, loops within conditionals, and so forth.

TIP Pay close attention to your loop's condition so that the loop ends at some point. Otherwise, you'll create an infinite loop, and the script will run and run and run.

Here's how to make the year pull-down menu using a **for** loop.

```

<select name="year">
<?php
$current_year = date('Y');
for ($i = $current_year; $i >=
($current_year - 100); $i--) {
echo "\n<option value=\"\$i\">
$i</option>";
}
?>
</select>

```