

VISUAL QUICKSTART GUIDE

**COVERS
PHP 5 & 7**



PHP for the Web

Fifth Edition

LARRY ULLMAN

© LEARN THE QUICK AND EASY WAY!

VISUAL QUICKSTART GUIDE

PHP for the Web

Fifth Edition

LARRY ULLMAN

Over the course of this chapter, a PHP script will be developed until it fully validates the **register.html** 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 completed. It's known as 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.

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

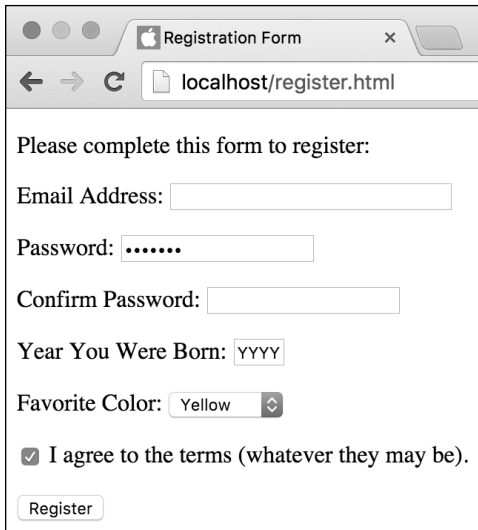
```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>Registration</title>
6  </head>
7  <body>
8    <h1>Registration Results</h1>
9    <?php // Script 6.2 - handle_reg.php
10   /* This script receives seven values
11    from register.html:
12    email, password, confirm, year, terms,
13    color, submit */
14
15   // Address error management, if you
16   want.
17
18   // Flag variable to track success:
19   $okay = true;
20
21   // If there were no errors, print a
22   success message:
23   if ($okay) {
24     print '<p>You have been
25     successfully registered (but not
26     really).</p>';
27   }
28   ?>
29 </body>
30 </html>
```

The variable is initialized with a Boolean value of *TRUE*, meaning that the assumption is that the form was completed properly. 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.



- 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.html**), and test both in your browser **B** and **C**.

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

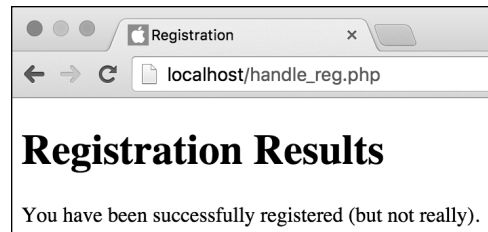
TIP If the statement area of your conditional is only one line long, you technically don't need the braces. In that case, you can write the conditional using either of these formats:

```
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 braces (as demonstrated in the syntax introduction) to improve consistency and minimize errors.



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

Validation Functions

PHP has dozens of functions commonly used to validate form data. Of these functions, three of the 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
```

The **isset()** function is commonly used to validate nontext form elements like checkboxes, radio buttons, and select menus. It's also regularly used to confirm that a variable exists, regardless of its value.

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
```

An interesting thing to note is that using **is_numeric()** on a variable that doesn't exist not only returns FALSE, but also generates a warning. For this reason, you'll often see **isset()** used along with other validation functions like **is_numeric()**.

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

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>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>Registration</title>
6      <style type="text/css"
7          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
15    from register.html:
16    email, password, confirm, year, terms,
17    color, submit */
18 // Address error management, if you
19    want.
20
21 // Flag variable to track success:
22 $okay = true;
23
24 // Validate the email address:
25 if (empty($_POST['email'])) {
26     print '<p class="error">Please
27         enter your email address.</p>';
28     $okay = false;
29 }
30
31 // Validate the password:
32 if (empty($_POST['password'])) {
33     print '<p class="error">Please
34         enter your password.</p>';
35     $okay = false;
36 }
37
38 // If there were no errors, print a
39    success message:
40 if ($okay) {
41     print '<p>You have been successfully
42         registered (but not really).</p>';
43 }
44 ?>
45 </body>
46 </html>

```

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;
}

```

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.

continues on next page

4. Repeat the validation for the password:

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

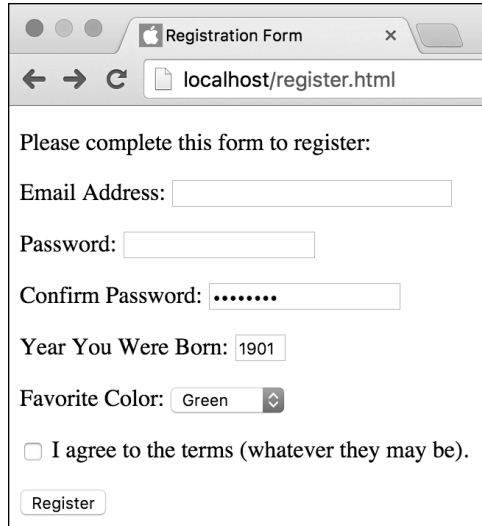
This is a repeat of the email validation, but with the variable name and **print** statement changed accordingly. The other form inputs will be validated in time.

All 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).

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

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

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



Registration Form

localhost/register.html

Please complete this form to register:

Email Address:

Password:

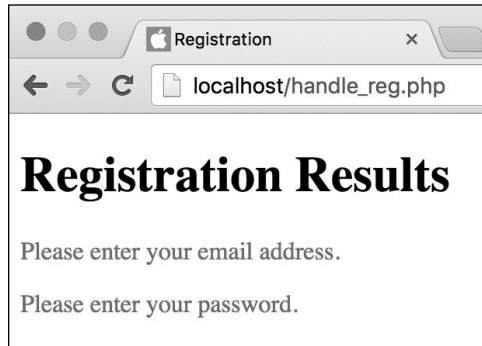
Confirm Password:

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

localhost/handle_reg.php

Registration Results

Please enter your email address.

Please enter your password.

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

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`.

TIP Once you're more comfortable with PHP, you'll start using the `filter()` function for validation, too. It's a wonderful tool, but a bit too complicated for beginners.