Programming with PHP

Ch2

Ed Crowley

Goals

At the end of this Chapter, you will be able to:

- Gather information with an HTML form.
- Use PHP to handle input from a form.
- Use conditionals and the remaining PHP operators…
- Employ concatenation, and mathematical operators, arrays, and loops.

Actions

- If you haven't already, download the complete set of textbook scripts and SQL commands from:

http://www.larryullman.com/downloads/phpmysql4_scripts.zip

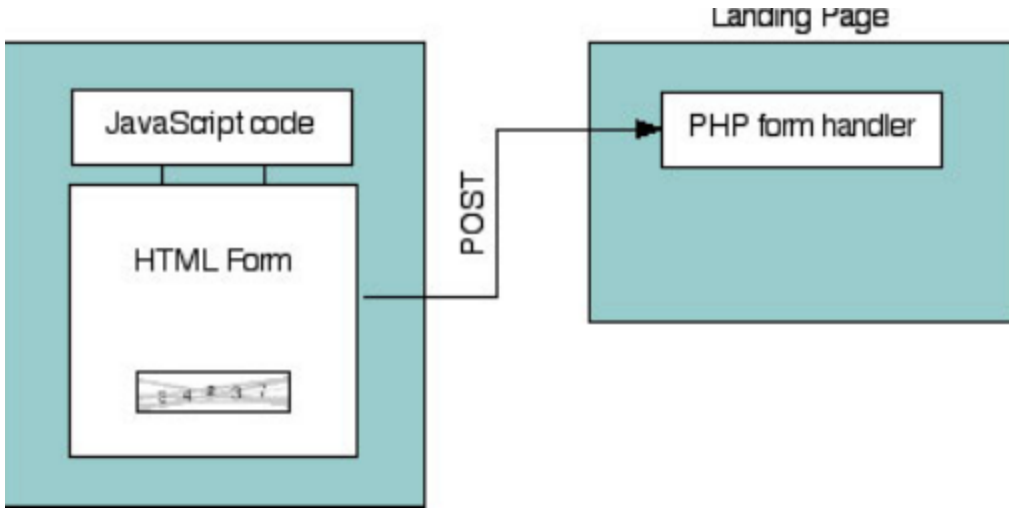
- Log into your Hostgator account.

HTML Forms

- In any dynamic Web site, handling an HTML form with PHP is arguably the most important process.

Two steps:

1. Create HTML form
2. Create corresponding PHP script to receive and process data from that form.



HTML Forms

- HTML forms are created using form tags and various input elements.

```
<form action="script.php" method="post">
Input elements here…
</form>
```

- Important form tag attribute is action
  - Dictates where form data is sent.
- Method attribute most frequently will be post.
- … different inputs—be they text boxes, radio buttons, select menus, check boxes, etc.—are placed within the opening and closing form tags.
- Remember names given to form inputs…

Script 2.1 Form.html

- To say that this script will be handling the form means that the PHP page will do something with the data it receives.
  - For now, our scripts will simply print the data back to the Web browser.
  - In a few weeks, we will be storing the input in a database.
- PHP scripts store the received information in special variables.
- For example:

```
<p><label>Name: <input type="text"
  name="name" size="20" maxlength="40"
  /></label></p>
```

Forms

- For example, say you have a form input like so:

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

- Whatever the user types into that input will be accessible via a PHP variable named:

```
$_REQUEST['city'].
```

- PHP is case-sensitive when it comes to variable names, so
  `$_REQUEST['city']` will work, but
  `$_Request['city']` and
  `$_REQUEST['City']` will have no value.

Script 2.2 Handle_form.php

- Next example will be a PHP script that handles the already-created HTML form from Script 2.1.
- Script will assign the form data to new variables.
- The script will then print the received values.
- Script 2.2 receives and prints out the information entered into an HTML form (Script 2.1).

```
// Create a shorthand for the form data:
$name = $_REQUEST['name'];
$comments = $_REQUEST['comments'];
*/// Print the submitted information:
echo "<p>Thank you, <b>$name</b>, for the
  following comments:<br
  /><tt>$comments</tt></p>
```

PHP's Three Primary Terms for Creating Conditionals

1) `if`
2) `else`
3) `elseif (can also be written as two words, else if)`

- Every conditional begins with an if clause:

```
if (condition) {
// Do something!
}
```

- An if can also have an else clause:

```
if (condition) {
// Do something!
} else {
// Do something else!
}
```

elseif

- An elseif clause allows you to add more conditions:

```
if (condition1) {
// Do something!
} elseif (condition2) {
// Do something else!
} else {
// Do something different!
}
```

- If a condition is true, the code in the following curly braces ({ }) will be executed.
- If not, PHP continues on.
- If there is a second condition (after an elseif), that will be checked.

elseif

- You can use as many elseif clauses as you want—until PHP hits an else, which will be automatically executed at that point, or until the conditional terminates without an else.
- It's important that the else always come last and be treated as the default action unless specific criteria—the conditions—are met. For example:

```
$x > $y
```

True

- A condition can be true in PHP for any number of reasons.
- For example, these are true conditions:
  - $var, if $var has a value other than 0, an empty string FALSE, or NULL
  - isset($var), if $var has any value other than NULL, including 0, FALSE, or an empty string

TABLE 2.2 Comparative and Logical Operators

Symbol	Meaning	Type	Example
==	is equal to	comparison	$x == $y
!=	is not equal to	comparison	$x != $y
<	less than	comparison	$x < $y
>	greater than	comparison	$x > $y
<=	less than or equal to	comparison	$x <= $y
>=	greater than or equal to	comparison	$x >= $y
!	not	logical	!$x
&&	and	logical	$x && $y
AND	and	logical	$x and $y
||	or	logical	$x || $y
OR	or	logical	$x or $y
XOR	and not	logical	$x XOR $y

To establish a True, or a False, condition, you can also use the comparative and logical operators (Table 2.2) in conjunction with parentheses to make more complicated expressions.

Script 2.3 handle_form.php Remade

- Two conditionals used to validate the gender radio buttons.
- Data submitted via HTML form should always be considered untrustworthy.

```
// Print a message based upon the gender value:
if ($gender == 'M') {
echo '<p><b>Good day, Sir!</b></p>';
}
 elseif ($gender == 'F') {
  echo '<p><b>Good day, Madam!</b></p>';
}  else {
// No gender selected
  echo '<p><b>You forgot to enter your
  gender!</b></p>';
}
```

Validating Form Data

- Validating form data requires the use of conditionals and any number of functions, operators, and expressions.
- One standard function to be used is isset( ), which tests if a variable has a value (including 0, FALSE, or an empty string, but not NULL).
- Preceding script was an example.

Empty()

- One issue with the isset( ) function is that an empty string tests as true, meaning that isset( ) is not an effective way to validate text inputs and text boxes from an HTML form.
- To check that a user typed something into textual elements, you can use the empty( ) function.
- It checks if a variable has an empty value: an empty string, 0, NULL, or FALSE.

Form Validation Goals

- First goal of form validation is seeing if something was entered or selected in form elements.
- Second goal is to ensure that submitted data is of the right type (numeric, string, etc.), of the right format (like an email address), or a specific acceptable value (like $gender being equal to either M or F).
- Let's create a new handle_form.php to make sure variables have values before they're referenced.

Script 2.4 Validating Form Data

- Validating HTML form data before you use it is critical to Web security and achieving professional results.
- Here, conditionals check that every referenced form element has a value.

```
// Validate the name:if
  (!empty($_REQUEST['name'])) {
  $name = $_REQUEST['name'];
} else {
  $name = NULL;
  echo '<p class="error">You forgot to
  enter your name!</p>';}
```

Arrays

- Unlike strings and numbers, an array can hold multiple, separate pieces of information.
  - An array is like a list of values, each value being a string or a number or even another array.
- Arrays are structured as a series of key-value pairs, where one pair is an item or element of that array.
  - For each item in the list, there is a key (or index) associated with it (Table 2.3).

PHP supports two kinds of arrays:

1. indexed, use numbers as the keys (as in Table 2.3)
2. associative, use strings as keys (Table 2.4).

Arrays: Indexed and Associative

TABLE 2.3 Array Example 1: `$artists`

Key	Value
0	The Mynabirds
1	Jeremy Messersmith
2	The Shins
3	Iron and Wine
4	Alexi Murdoch

TABLE 2.4 Array Example 2: `$states`

Key	Value
MD	Maryland
PA	Pennsylvania
IL	Illinois
MO	Missouri
IA	Iowa

Arrays

- Indexed arrays begin with the first index at 0.
  - Unless you specify keys explicitly.
- An array follows the same naming rules as any other variable.
  - Means that you might not be able to tell that $var is an array as opposed to a string or number.
- To refer to a specific value in an array, start with the array variable name, followed by the key within square brackets:

```
$band = $artists[0]; // The Mynabirds
echo $states['MD']; // Maryland
```

- You can see that the array keys are used like other values in PHP: numbers (e.g., 0) are never quoted, whereas strings (MD) must be.

Array Syntax

- Because arrays use a different syntax than other variables, and can contain multiple values, printing them can be trickier.
- This will not work:

```
echo "My list of states: $states";
```

- However, printing an individual element's value is simple if it uses indexed (numeric) keys:

```
echo "The first artist is $artists[0].";
```

- But if the array uses strings for the keys, the quotes used to surround the key will muddle the syntax.

Array Names

- The following code will cause a parse error:

```
echo "IL is $states['IL]."; // BAD!
```

- To fix this, wrap the array name and key in curly braces when an array uses strings for its keys:

```
echo "IL is {$states['IL']}.";
```

Script 2.5 -- handle_form v4

- You've already worked with two arrays: $_SERVER (in Chapter 1) and $_REQUEST (in this chapter).
- To acquaint you with another array and to practice printing array values directly, one final, but basic, version of the handle_form.php page will be created using the more specific $_POST array.
- The superglobal variables, like $_POST here, are just one type of PHP array.

Superglobal Arrays

- PHP includes several predefined arrays called the superglobal variables. They are:

`$_GET, $_POST, $_REQUEST, $_SERVER, $_ENV, $_SESSION, and $_COOKIE.`

- $_GET variable is where PHP stores all of the values sent to a PHP script via the GET method.
- $_POST stores all of the data sent to a PHP script from an HTML form that uses the POST method.
- Both of these—along with $_COOKIE—are subsets of $_REQUEST, which you've been using.
- $_SERVER, which was used in Chapter 1, stores information about the server PHP is running on, as does $_ENV. $_SESSION.

Creating arrays

- Frequently there will be times when you want to create your own array.
- Two primary ways to define an array.
- First, add an element at a time to build one:

```
$band[] = 'Jemaine';
$band[] = 'Bret';
$band[] = 'Murray';
```

- As arrays are indexed starting at 0, $band[0] has a value of Jemaine; $band[1], Bret, and $band[2], Murray.

Associative Arrays

- Alternatively, specify the key when adding an element.
- But it's important to understand that if you specify a key and a value already exists indexed with that same key, the new value will overwrite the existing one:

```
$band['fan'] = 'Mel';
$band['fan'] = 'Dave'; // New value
$fruit[2] = 'apple';
$fruit[2] = 'orange'; // New value
```

Array() function

- You can use the array( ) function to build an entire array in one step:

```
$states = array ('IA' => 'Iowa' ,'MD' =>
  'Maryland');
```

- The array( ) function can be used whether or not you explicitly set the key:

```
$artists = array ('Clem Snide', 'Shins'
  'Eels');
```

- Or, if you set the first numeric key value, the added values will be keyed incrementally thereafter:

```
$days = array (1 => 'Sun' 'Mon',0,'Tue');
echo $days[3]; // Tue
```

array() function

- The array( ) function is also used to initialize an array, prior to referencing it:

```
$tv = array( );
$tv[] = 'Flight of the Conchords';
```

Accessing Entire Arrays

- To access every array element, use the foreach loop:

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

- The foreach loop will iterate through every element in $array, assigning each element's value to the $value variable.
- To access both the keys and values, use:

```
 foreach ($array as $key => $value) {
echo "The value at $key is $value.";
}
```

- (You can use any valid variable name in place of $key and $value, like just $k and $v.)

Script 2.6

- Using arrays, this next script will demonstrate how easy it is to make a set of form pull-down menus for selecting a date
- This form uses arrays to dynamically create three pull-down menus.

```
// Make the days pull-down menu:
  echo '<select name="day">';
  foreach ($days as $value) {
    echo "<option value=\"$value\">$value</option>\n";
   }
   echo '</select>';
```

Multidimensional arrays

- Multidimensional arrays remarkably easy to work with.
- As an example, start with an array of prime numbers:

```
$primes = array(2, 3, 5, 7, ...);
```

- Then create an array of sphenic numbers (don't worry: I had no idea what a sphenic number was either; I had to look it up):

```
$sphenic = array(30, 42, 66, 70, ...);
```

- These two arrays could be combined into one multidimensional array like so:

```
$numbers = array ('Primes' => $primes,
  'Sphenic' => $sphenic);
```

- Now, $numbers is a multidimensional array.

Accessing Multidimension Arrays

- To access the prime numbers sub-array, refer to $numbers['Primes'].
- To access the prime number 5, use $numbers['Primes'][2] (it's the third element in the array, but the array starts indexing at 0).
- To print out one of these values, surround the whole construct in curly braces:

```
echo "The first sphenic number is
  {$numbers['Sphenic'][0]}.";
```

- Of course, you can also access multi-dimensional arrays using the foreach loop, nesting one inside another if necessary.

Script 2.7 Multi.php

- Multidimensional arrays are created by using other arrays for its values.
  - Two foreach loops, one nested inside of the other, can access every array element.

```
// Loop through the countries:
foreach ($n_america as $country => $list) {
  // Print a heading:
  echo "<h2>$country</h2><ul>";
  // Print each state, province, or
  territory:
  foreach ($list as $k => $v) {
      echo "<li>$k - $v</li>\n";
  }
  // Close the list:
  echo '</ul>';
} // End of main FOREACH.
```

Sorting Arrays

- PHP includes several functions you can use for sorting arrays:

```
$names = array ('Moe' 'Larry',, 'Curly');
sort($names);
```

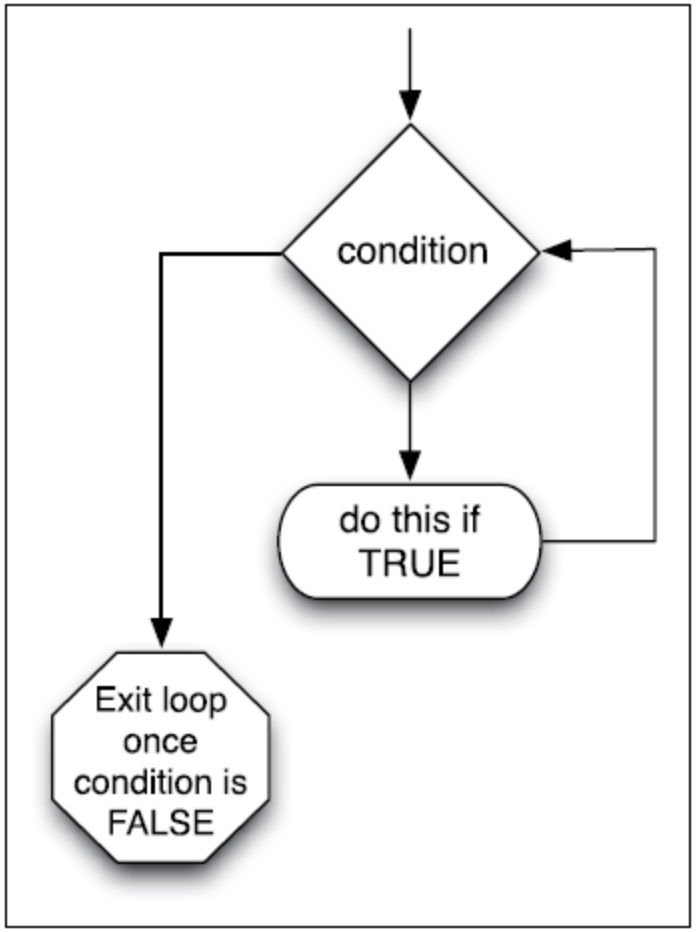
The sorting functions perform three kinds of sorts.

- First, you can sort an array by value, discarding the original keys, using sort( ).
  - It's important to understand that the array's keys will be reset after the sorting process, so if the key-value relationship is important, you should not use sort( ).
- Second, you can sort an array by value while maintaining the keys, using asort( ).
- Third, you can sort an array by key, using ksort( ).

Script 2.8

- Each of these can sort in reverse order if you change them to rsort( ), arsort( ), and krsort( ) respectively.
- To demonstrate the effect sorting arrays will have, this next script will create an array of movie titles and ratings (how much I liked them on a scale of 1 to 10) and then display this list in different ways.

- In Script 2.8, an array is defined, then sorted in two different ways: first by key, then by value (in reverse order).



While Loop

- while loop looks like this:

```
while (condition) {
// Do something.
}
```

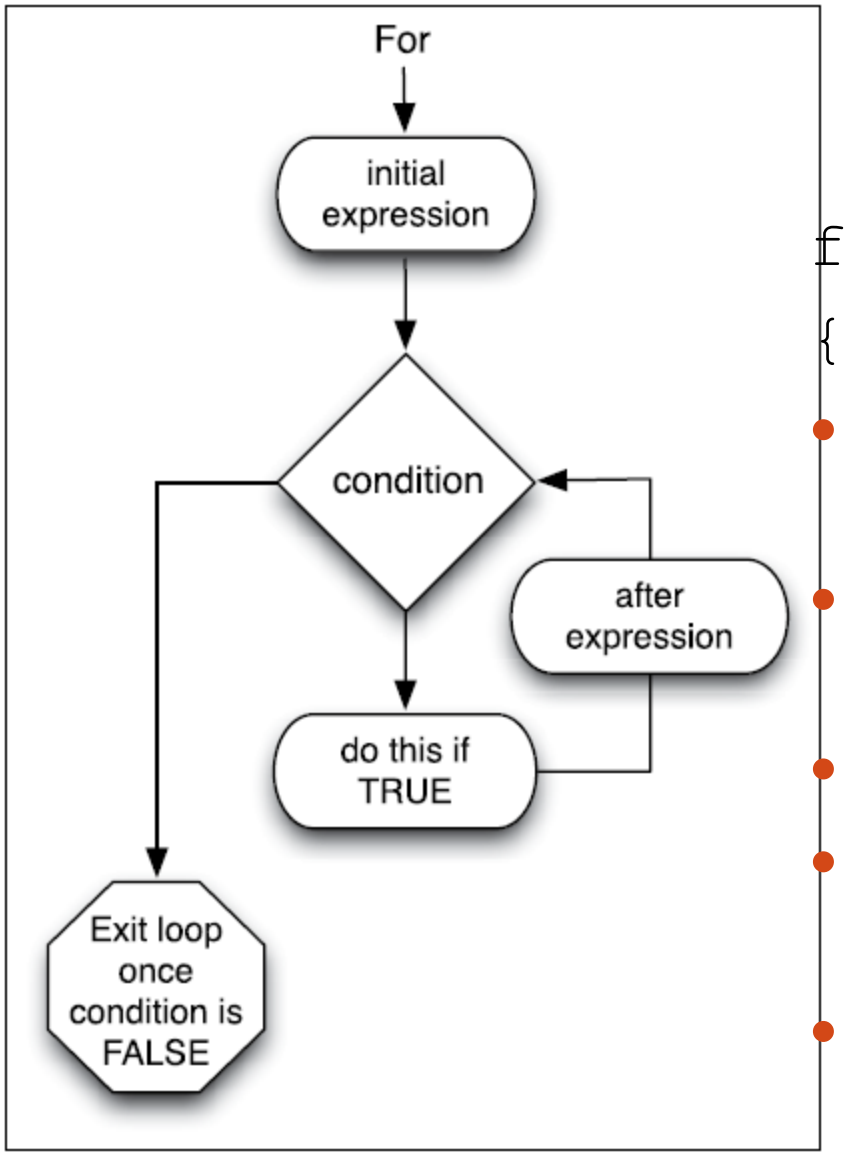
- As long as the condition part of the loop is true, the loop will be executed.
- Once it becomes false, the loop is stopped
- If the condition is never true, the loop will never be executed.
- while loop most frequently used when retrieving results from a database, ."

For loop

- A more complicated syntax:

```
for (initial expression; condition;
closing expression) {
// Do something.
}
```

- Upon first executing the loop, the initial expression is run.
- Then the condition is checked and, if true, the contents of the loop are executed.
- After execution, the closing expression is run and the condition is checked again.
- Process continues until the condition is false



For Loop Example

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

- First time this loop is run, the $i variable is set to the value of 1.
- Then condition is checked (is 1 less than or equal to 10?).
- Since this is true, 1 is printed out (echo $i).
- Then, $i is incremented to 2 ($i+ +), the condition is checked, and so forth.
- Result will be numbers 1 through 10 printed out.

Script 2.9

- The functionality of both loops is similar enough that for and while can often be used interchangeably.
- Still, for loop is a better choice for doing something a known number of times…
- whereas while is used when a condition will be true an unknown number of times.
- In this chapter's last example, the calendar script created earlier will be rewritten using for loops in place of two of the foreach loops.
- Loops often used in conjunction with or in lieu of an array. Here, two for loops replace the arrays and foreach loops used in the script previously.

Questions???