

Introduction to PHP

Lesson 1: **Introduction**

[Working in CodeRunner](#)

[Creating a File](#)

[Managing your Files](#)

[Four characteristics of PHP](#)

[1. PHP is a server-side language, with HTML embedding.](#)

[2. PHP is a Parsed language.](#)

[3. PHP works jointly with SQL.](#)

[4. PHP is part of the LAMP, WAMP, and MAMP stack.](#)

Lesson 2: **PHP Basics**

[PHP Delimiters and Comments](#)

[Variables in PHP](#)

[Modifying Variables and Values with Operators](#)

[Superglobals](#)

[\\$GLOBALS](#)

[\\$ _SERVER:](#)

[\\$ _GET](#)

[\\$ _POST](#)

Lesson 3: **Decisions**

[Comparison Operators and Conditions](#)

[IF and ELSE Control Structure](#)

[Logical Operators](#)

[A Brief Preview of Forms](#)

Lesson 4: **Multiple Control Structures and Loops**

[Multiple Control Structures](#)

[WHILE and FOR Loops](#)

Lesson 5: **Functions**

[Creating Code Reusability with Functions](#)

[Function and Variable Scopes](#)

[Using Functions with Parameters and Return Values](#)

[Sneaking In with Parameters](#)

[Sneaking out with Return Values](#)

[Multiple Parameters and Default Values](#)

Lesson 6: **Arrays**

[Creating an Array](#)

[Associative Arrays](#)

[Creating Multi-Dimensional Arrays](#)

[Traversing and Manipulating Arrays](#)

[Traversing Associative Arrays with list\(\) and each\(\)](#)

[More built-in functions](#)

Lesson 7: **Strings**

[What's a String Anyway?](#)

[Manipulating Strings](#)

[Other nifty string shortcuts](#)

[Built-in String Functions](#)

[Regular Expressions](#)

[Character Ranges and Number of Occurrences](#)

[Excluding Characters](#)

[Escaping Characters](#)

Lesson 8: **Fixing Broken PHP**

[Things Professors Don't Talk About Enough](#)

[Debugging Tips](#)

[Utilizing Error Messages](#)

[Riddle-Me-This Error Messages](#)

[Errors without Error Messages](#)

[Logical Errors](#)

[Infinite Loops, Infinite Headaches](#)

[Notes on Scalable Programming](#)

[Before you Code, Pseudocode](#)

[Make your Program Readable](#)

[Comment Until You're Blue in the Face](#)

[Code in Bite-Size Chunks](#)

[Debug as You Work](#)

[Reuse Functions as Much as Possible](#)

[Utilize Available Resources](#)

Lesson 9: **Forms in PHP**

[Forms Review](#)

[Using Superglobals to Read Form Inputs](#)

[Extracting Superglobals into Variables](#)

[Nesting Variable Names](#)

Lesson 10: **Utilizing Internet Tools**

[Environment and Server Variables](#)

[Using HTTP Headers](#)

[Manipulating Query Strings](#)

[Customizing specific error messages](#)

[Sending Emails](#)

Lesson 11: **Date and Time**

[Date and Time Standards](#)

[Date and Time Functions](#)

[Constructing Dates and Times](#)

Lesson 12: **Using Files**

[Including and Requiring Files](#)

[Reading and Writing Files](#)

Allowing Users to Download Files

Lesson 13: **Cookies and Sessions**

Using Cookies

Knowing the User Through Sessions

Deleting Sessions

Lesson 14: **Final Project**

Final Project

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Introduction

Welcome to the O'Reilly School of Technology **Introduction to PHP** course!

In this PHP class, you will learn basic to intermediate programming aspects of PHP--hypertext preprocessor. PHP is a versatile server-side programming language that works hand-in-hand with front-end web languages such as HTML and JavaScript. PHP can be used to create all types of dynamic web interfaces, and because of its open-source robustness, has become one of the most widely used programming languages for the internet.

Course Objectives

When you complete this course, you will be able to:

- develop web applications using basic PHP elements such as delimiters, control structures, operators, variables, arrays, and functions.
- manipulate dates and strings using built-in PHP functions and regular expressions.
- debug and improve code for better reusability and scalability.
- create dynamic web forms using internet tools such as input, environment and server variables, HTTP headers, and query strings.
- read, write, manage and download files through PHP-based web applications.
- track user information using cookies and sessions.
- build a full-fledged shopping cart system.

From beginning to end, you will learn by doing your own PHP based projects. These projects, as well as the final project, will add to your portfolio and provide needed experience. Besides a browser and internet connection, all software is provided online by the O'Reilly School of Technology.

Learning with O'Reilly School of Technology Courses

As with every O'Reilly School of Technology course, we'll take a *user-active* approach to learning. This means that you (the user) will be active! You'll learn by doing, building live programs, testing them and experimenting with them—hands-on!

To learn a new skill or technology, you have to experiment. The more you experiment, the more you learn. Our system is designed to maximize experimentation and help you *learn to learn* a new skill.

We'll program as much as possible to be sure that the principles sink in and stay with you.

Each time we discuss a new concept, you'll put it into code and see what YOU can do with it. On occasion we'll even give you code that doesn't work, so you can see common mistakes and how to recover from them. Making mistakes is actually another good way to learn.

Above all, we want to help you to *learn to learn*. We give you the tools to take control of your own learning experience.

When you complete an OST course, you know the subject matter, *and* you know how to expand your knowledge, so you can handle changes like software and operating system updates.

Here are some tips for using O'Reilly School of Technology courses effectively:

- **Type the code.** Resist the temptation to cut and paste the example code we give you. Typing the code actually gives you a feel for the programming task. Then play around with the examples to find out what else you can make them do, and to check your understanding. It's highly unlikely you'll break anything by experimentation. If you *do* break something, that's an indication to us that we need to improve our system!
- **Take your time.** Learning takes time. Rushing can have negative effects on your progress. Slow down and let your brain absorb the new information thoroughly. Taking your time helps to maintain a relaxed, positive approach. It also gives you the chance to try new things and learn more than you otherwise would if you blew through all of the coursework too quickly.
- **Experiment.** Wander from the path often and explore the possibilities. We can't anticipate all of your questions and ideas, so it's up to you to experiment and create on your own. Your instructor will help if you go completely off the rails.
- **Accept guidance, but don't depend on it.** Try to solve problems on your own. Going from misunderstanding to understanding is the best way to acquire a new skill. Part of what you're learning is problem solving. Of course, you can always contact your instructor for hints when you need them.

- **Use all available resources!** In real-life problem-solving, you aren't bound by false limitations; in OST courses, you are free to use any resources at your disposal to solve problems you encounter: the Internet, reference books, and online help are all fair game.
- **Have fun!** Relax, keep practicing, and don't be afraid to make mistakes! Your instructor will keep you at it until you've mastered the skill. We want you to get that satisfied, "I'm so cool! I did it!" feeling. And you'll have some projects to show off when you're done.

Lesson Format

We'll try out lots of examples in each lesson. We'll have you write code, look at code, and edit existing code. The code will be presented in boxes that will indicate what needs to be done to the code inside.

Whenever you see white boxes like the one below, you'll *type* the contents into the editor window to try the example yourself. The CODE TO TYPE bar on top of the white box contains directions for you to follow:

CODE TO TYPE:

White boxes like this contain code for you to try out (type into a file to run).
If you have already written some of the code, new code for you to add looks like this.
If we want you to remove existing code, the code to remove ~~will look like this~~.
We may also include instructive comments that you don't need to type.

We may run programs and do some other activities in a terminal session in the operating system or other command-line environment. These will be shown like this:

INTERACTIVE SESSION:

The plain black text that we present in these INTERACTIVE boxes is provided by the system (not for you to type). The commands we want you to type look like this.

Code and information presented in a gray OBSERVE box is for you to *inspect* and *absorb*. This information is often color-coded, and followed by text explaining the code in detail:

OBSERVE:

Gray "Observe" boxes like this contain **information** (usually code specifics) for you to observe.

The paragraph(s) that follow may provide addition details on **information** that was highlighted in the Observe box.

We'll also set especially pertinent information apart in "Note" boxes:

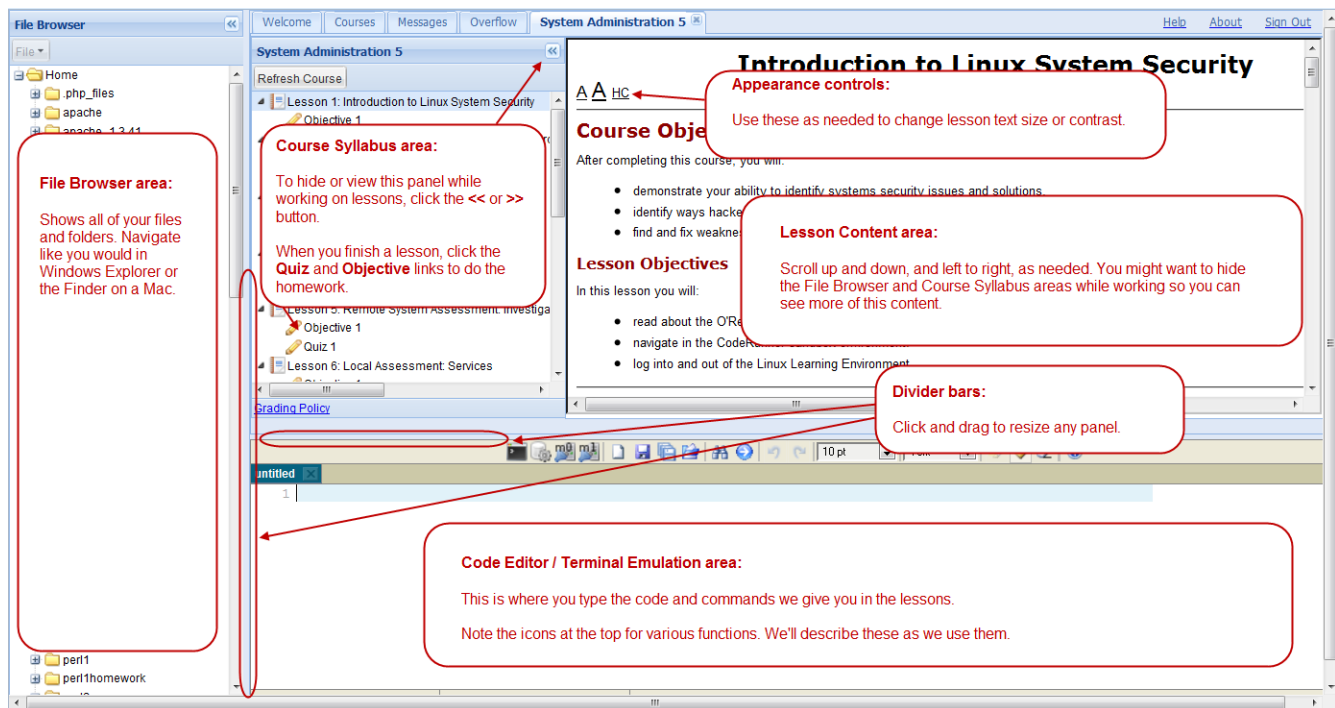
Note Notes provide information that is useful, but not absolutely necessary for performing the tasks at hand.

Tip Tips provide information that might help make the tools easier for you to use, such as shortcut keys.

WARNING Warnings provide information that can help prevent program crashes and data loss.

The CodeRunner Screen

This course is presented in CodeRunner, OST's self-contained environment. We'll discuss the details later, but here's a quick overview of the various areas of the screen:



These videos explain how to use CodeRunner:

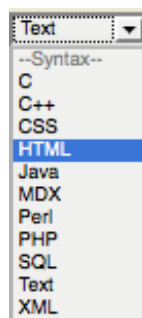
[File Management Demo](#)

[Code Editor Demo](#)

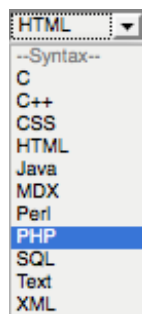
[Coursework Demo](#)

Working in CodeRunner

Since CodeRunner is a multi-purpose editor, you need to make sure you're using the correct **syntax**. In this course, you will be using HTML and PHP. To start using HTML, choose the **HTML** option:



To change to PHP, choose the PHP option:



Creating a File

Let's create a file now. Select the HTML syntax and type the code as shown below.

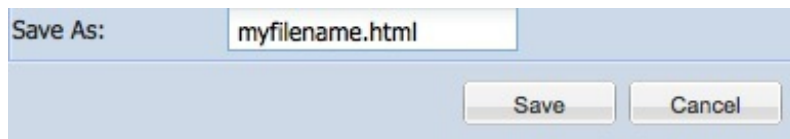
Make sure you're using HTML syntax and type the following into CodeRunner:


```
Four characteristics of PHP:

<ol>
  <li> PHP is a server-side language with HTML embedding.</li>
  <li> PHP is a parsed language.</li>
  <li> PHP works hand-in-hand with SQL.</li>
  <li> PHP is part of the LAMP stack.</li>
</ol>
```

Managing your Files

Click the  button. In the Save As text box, type **fourfacts.html** (be sure to include the html extension when you Save html files).




You can also use the **Save As** () button to save a file with a different name. Try it now with the name **fourfacts2.html**. Note that you are now editing fourfacts2.html, not fourfacts.html.

After you successfully save your file, anybody can go on the web, type the URL (<http://yourusername.oreillystudent.com/fourfacts.html>) in the location bar of their browser, and see this page.

To retrieve the original **fourfacts.html**, click the **Load File** () button or double-click the file name in the **File Browser** window.

Four characteristics of PHP

Look again at the HTML top-four list you just typed into CodeRunner, and click Preview:

Note Keep in mind that every time you Preview a file, your changes will be saved. Think about whether you want the previous code overwritten or not. If not, use Save As  before you Preview.

Four characteristics of PHP:

1. PHP is a server-side language with HTML embedding.
2. PHP is a parsed language.
3. PHP works hand-in-hand with SQL.
4. PHP is part of the LAMP stack.



Note If the Preview button doesn't work for you, you may be blocking pop-up windows in your browser. To fix this, change your configuration settings to allow pop-ups from the OST servers, or view your page directly at <http://yourusername.oreillystudent.com/fourfacts.html>.

This example serves more than one purpose for us. It demonstrates how to use CodeRunner and it introduces some keys to using PHP. Of course there's much more to PHP than this, but let's start with this.

1. PHP is a server-side language, with HTML embedding.

On the web there are two sides to everything: the Client Side and the Server Side. The Client side is the side you are on right now. It consists of your computer and your web browser. The server side is the side where the web pages are stored and where programs are executed to build dynamic web pages with PHP.

Still have your HTML list? It's time to convert it to PHP. **Switch CodeRunner to PHP**, and retype the top four list into the editor. Then add the blue code below:

Make sure you're using PHP then type the following into CodeRunner:

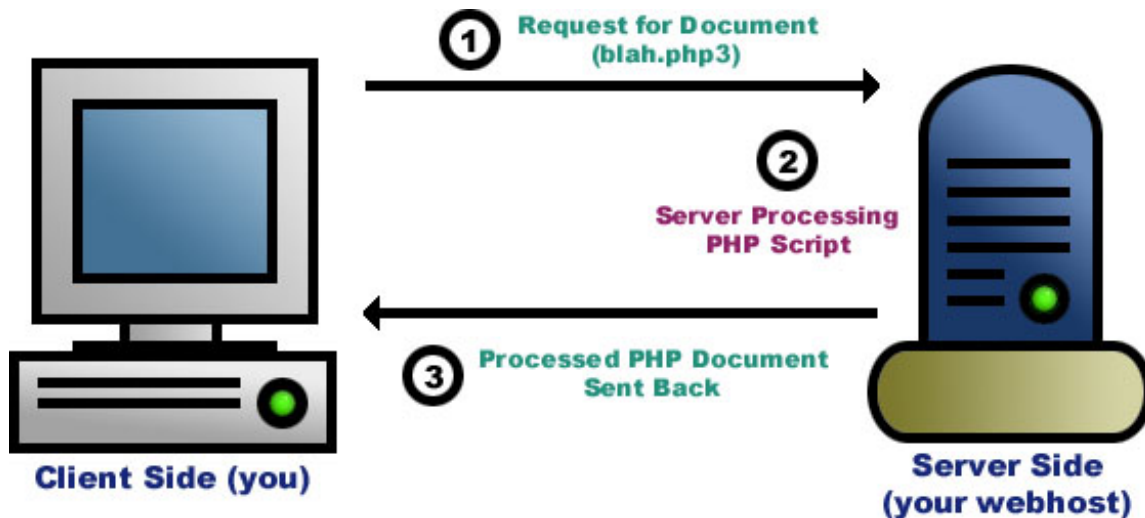
```
<?php echo "Four characteristics of PHP: "; ?>

<ol>
<li> PHP is a server-side language with HTML embedding. </li>
<li> PHP is a parsed language. </li>
<li> PHP works hand-in-hand with SQL. </li>
<li> PHP is part of the LAMP stack. </li>
</ol>
```

Click **Preview**. This time save with the php (.php) extension. It looks exactly the same, right? But something more happened this time on the back end.

You see, HTML is a *Client side* language. When you clicked **Preview** while in **HTML**, the Sandbox simply asked your browser to process the HTML tags without any outside help.

Conversely, PHP is a *server side* scripting language and builds HTML dynamically before sending to your browser. Here's a diagram of how PHP works:



When you used **Preview** after adding the PHP code while using **PHP** syntax, the Learning Sandbox:

- Took your code back to your Lab Account on our web server
- *Parsed* it using the **PHP Engine** that's installed within your account
- Returned the results to the browser as HTML

Then your browser rendered the HTML to make it look pretty. Did you notice how the addition of PHP code at the top of the file did nothing to change the HTML list below? This is because the HTML is *embedded* into the PHP file, and doesn't require anything else to output it.

2. PHP is a Parsed language.

The fact that PHP is a parsed language as opposed to a compiled language is a technical concern and probably only interesting to programmers with experience in *Compiled* programming languages like Java or C++. Those languages perform an additional task called compiling that turns the text from the program into a form the computer understands. A binary file is created that serves as the thing that gets executed when a program is running.

PHP is a **Parsed** language, meaning that you can see the results of your code immediately after saving the file, without any compiling or linking steps in between. That's because the compiled PHP engine installed on your account takes the PHP file you've created and "parses" it and uses the commands you created to make the server do something. All the work is still done by a compiled program, but the program you created doesn't have to be compiled, since it just tells the compiled program what to do.

For the geeks out there, this is similar to an **Interpreted** language such as Perl; however, the parsing process has been optimized to use a combination of interpreting and compiling at **run-time**, enabling PHP to be powerful AND fast.

The bottom line is the parsing action of PHP makes your life easier. If you want to know more about parsed, interpreted, and compiled languages, here's a good [link](#).

3. PHP works jointly with SQL.

Let's look at your first PHP script again and add one more little piece of code. Don't worry yet about what the code means, at this point we're just playing around.

Type the following (in BLUE) into CodeRunner:

```
<?php echo "Four facts about PHP:"; ?>
<ol style="font-size:16px;">
<li> PHP is a server-side language with HTML embedding. </li>
<li> PHP is a parsed language. </li>
<li> PHP works hand-in-hand with SQL. </li>
<? printf("MySQL client info: %s\n", mysqli_get_client_info()); ?>
<li> PHP is part of the LAMP stack. </li>
</ol>
```

Click **Preview**. Now you should see the version of **MySQL** library that's included with your account's PHP engine, embedded within your HTML list. You'll learn a lot more about the MySQL database in later courses, but for now just roll with it.

PHP makes it easy to add database-driven content to any website. It supports popular database systems - MySQL, PostgreSQL, Oracle, and others - with libraries of built-in **functions** like the one you added above. These libraries can be referred to by the acronym **DBI**: Database Interface.

Other programming languages such as Perl contain their own sets of DBI libraries too. However, unlike Perl, PHP was designed with database-driven websites in mind, and has become so closely intertwined with MySQL that the two organizations now work together to ensure continued reciprocal support.

Here's a good [O'Reilly article](#) about PHP and MySQL.

4. PHP is part of the LAMP, WAMP, and MAMP stack.

What's a (L|W|M)AMP Stack?

It's yet another acronym.

- **L**inux, **W**indows, **M**ac
- **A**pache
- **M**ySQL
- **P**HP (or **P**erl, or both)

The **Stack** part refers to a group of technologies which, when used together, create powerful and dynamic web applications. There are competing stacks, such as Microsoft's .NET framework and Sun's Java/J2EE technologies. However, corporations are realizing more and more that the free, open-source LAMP Stack can be just as powerful, safe, and lucrative for their businesses as the expensive, proprietary competitors.

And by the way, lucky you! You have all the LAMP technologies you need at your fingertips *RIGHT NOW*:

- Your Learning Lab account is on a **Linux RedHat** server.
- It's equipped with its own **Apache** web server.
- The Apache server has **MySQL** installed on it.
- It also has **PHP AND Perl** on it.

Alright, you're doing great so far! Don't forget to **Save** your first PHP file (call it "**first.php**"), and work on this lesson's assignments on the syllabus page. Be sure to read the comments on each project or quiz using the "Graded" link. See you in the next lesson!



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

PHP Basics

Welcome back! In the next three lessons, we'll be playing around with a make-believe program to help demonstrate a few programming concepts. While the faux-program isn't one you'd likely create while on the job, the concepts and techniques used are the same. Let's get started!

PHP Delimiters and Comments

Open the file **first.php**. Or, if you're completely sick of the Top Four list, start a new file. Make sure you're using PHP.

Type the following green and blue code into your chosen file in CodeRunner:

```
<ol style="font-size:16px;">
  <li>
    PHP is a server-side language, with HTML embedding.
  </li><br/>
  For instance:<br>
  <ul>
    <?php
      echo "<li style='color:blue;'"
        This PHP code is INSIDE the PHP delimiters
      </li>";
    ?>
    <li style="color:green;">
      This HTML code is OUTSIDE the PHP delimiters
    </li>
  </ul>
</ol>
```

Now click **Preview** to see the results. What happened?

It should look something like this:

1. PHP is a server-side language, with HTML embedding.
For instance:
 - This PHP code is INSIDE the PHP delimiters.
 - This HTML code is OUTSIDE the PHP delimiters.



PHP code is separated from embedded **HTML** with **delimiters**. The delimiters are the **<?php** and the **?>**. All PHP is written between these delimiters. An open delimiter **<?php** must have a closing delimiter **?>**. Try taking the delimiters away and see what happens.

Take a look at the PHP - notice that between the delimiters (**<?php** and **?>**), the word **echo** showed up. That's a PHP command that means "make this show up." Without echo, it won't show up. Remove the echo command and check out the results.

Here the **echo statement** is used to return text to the web browser from a PHP script. The "echo" command just means "print this out." And one more thing, all statements in PHP must end with a semicolon (;).

By the way, why did we put in those slashes (**//**) in the sample code above? Well, those are called "comments." Commenting is a common practice in programming. Commenting records the specific reasons you had for writing the program a particular way.

Type the blue text below into CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is a server-side language, with HTML embedding.</li>
  <br/> For instance: <ul>

    <?php
      //Jerry says, "What's the deal with this line of PHP code?"
      /* Elaine says, "I want to talk about this line, that line, AND the other line!"
    */
    #George says, "...yadda yadda yadda..."
    echo "<li style='color:blue;'>
      This PHP code is INSIDE the PHP delimiters
    </li>";
  ?>
  <li style="color:green;">
    This HTML code is OUTSIDE the PHP delimiters
  </li>
</ul>
</ol>
```

Now click **Preview** to see the results. Notice what ISN'T printed out on the screen. Our results look exactly the same as before.

What happened to the entire conversation we added?

Ah, Jerry, Elaine, and George, always commenting on everything, yet doing pretty much nothing. In fact, they behave a lot like **comments** in PHP, but in PHP, it's a developer who comments on the code without impacting the results at all.

Note

Two slashes (*//*) or one pound sign (*#*) will "comment out" the line that follows it. Or, you can comment out multiple lines by surrounding them with */** and **/*. Play around a bit to get the hang of it.

Comments may not seem like a big deal (they didn't to me at first either), but as our programs become more complicated, it's useful to have reminders (in your own words) of the specific reasons you chose to write your code a certain way. Also, comments are essential for reusing and sharing code. They allow other developers to decipher and understand the specifics of the code you've written.

Let's experiment to discover the answers to these questions:

- Can **delimiters** share the same line as the PHP code?
- Can they share the same line as HTML?
- Can multiple **statements** share *one* line?
- Can **statements** span *multiple* lines?
- Does it matter whether you use *<?php* or just *<? ?*
- Can you mix a line of code with a comment?

Variables in PHP

Every programming language has **variables**. Variables are places to store things. You can insert values into variables and you can extract them. Variables are a lot like dresser drawers. You can put things in and then take them out, and you usually know what each drawer contains.

Add the following blue text to your file in CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is part of the LAMP stack.</li>
  <?php
    $lamp_l = "Linux";
    $lamp_a = "Apache";
    $lamp_m = "MySQL";
    $lamp_p = "PHP";
  ?>
</ol>
```

Now Preview that.

Not much happening, right? That's because we forgot to *echo* something. Let's go ahead and do that so we get an output.

Add the following blue text to your file in CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is part of the LAMP stack.</li>
  <?php
    $lamp_l = "Linux";
    $lamp_a = "Apache";
    $lamp_m = "MySQL";
    $lamp_p = "PHP";
    echo $lamp_a;
  ?>
</ol>
```

Now you should see the word **Apache** printed on your page. Why do you suppose that is? After all, we wrote **Echo \$lamp_a;**, you'd think that would be the word that printed. Well, as it turns out, **\$lamp_a** is a variable. Variables in PHP always begin with a \$. And, as if **\$lamp_a** was a drawer, we put something in it, we inserted **= "Apache"**. Then, in order to get something out of the variable **\$lamp_a**, we "echoed" by adding **echo \$lamp_a**. Finally, "Apache," the value we put into the variable (or drawer) in the first place was printed out. They are called "variables" because the value can *vary*. We can change the contents of the variable at anytime, and that makes them very useful for storing and retrieving values dynamically.

Just like you can put different items in the appropriate drawers of your dresser--you might have a sock drawer for your socks, a shirt drawer for your shirts, etc.--you can put different kinds of values in variables. In the example above, we've entered words into our variables. In programming, these words are called **strings** because they comprise a string of letters or characters.

Let's put different kinds of values into some other variables.

Add the following blue text to your file in CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is part of the LAMP stack.</li>
  <?php
    $lamp_l = "Linux"; $lamp_a = "Apache"; $lamp_m = "MySQL"; $lamp_p = "PHP";
  ?>
  /* Here are some imaginary numbers for a possible salary package associated with the
  skills we're learning in this course (play along!): */
  <?php
    $base_salary = 158470;
    //whoa. we hit the jackpot
    $bonus = 25815.25;
    $benefits = 0.2;
    //percentage of total
    $time_off = 6476; //in dollars
  ?>
</ol>
```

Here we're *assigning* different kinds of **values** (like **158470**) to **variables**. In this context, *assigning* simply means to "fill" the variable with a value.

We've *assigned* the various PHP variables values of three basic **types**: **integer**, **floating point (decimal)**, and **string**. Integers are whole numbers, including negative numbers and 0. Floating point numbers are numbers that may have a decimal point. Strings, as we already mentioned, are simply successive *strings* of characters.

Below is a list of the **types** we've assigned to the variables listed:

Variable Name	Assigned Value	Value Type
\$lamp_l	"Linux"	string
\$lamp_a	"Apache"	string
\$lamp_m	"MySQL"	string
\$lamp_p	"PHP"	string
\$base_salary	158470	integer
\$bonus	25815.25	float
\$benefits	0.2	float
\$time_off	6476	integer

Let's go back to our dresser analogy for a moment. It will help us to understand the difference between "strongly typed" programming languages and languages like PHP, which are not strongly typed. Programming languages that are *strongly typed* require you to decide the types of variables you're going to have upon creating your files. Once you've created your variables, you are committed. They cannot be revised. It's like labeling your drawers, so that you can only put socks in your sock labeled drawer and shirts in your shirt labeled drawer. And after you've labeled these drawers, you can't change them. PHP, however, is not strongly typed. Therefore, the variables remain flexible, we are allowed to change them, and we can put any type of information into a PHP file that we wish.

Since PHP is NOT a **strongly typed** programming language, the following won't break your script:

Type the following blue text below into CodeRunner:

```
<ol style="font-size:16px;">
<li> PHP is part of the LAMP stack. </li>
  <?php
    $lamp_l = "Linux";
    $lamp_a = "Apache";
    $lamp_m = "MySQL";
    $lamp_p = "PHP";
  ?>
  <?php
    /* Here are some imaginary numbers for a possible salary package for the skills we're learning: */
    $base_salary = 158470;
    $bonus = 25815.25;
    $benefits = 0.2;
    //percentage of total
    $time_off = 6476;
    //in dollars
    $benefits = "this is a string now"; // You just changed the value of $benefits from a float number to a string!
  ?>
</ol>
```

Go ahead and Preview it. Our variables are shy creatures! So far they've been hiding like comments when we **Preview**. Let's add some HTML and echo out some of those variables.

Type the following blue and green text into your document in CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is part of the LAMP stack. </li>
  <?php
    $lamp_l = "Linux";
    $lamp_a = "Apache";
    $lamp_m = "MySQL";
    $lamp_p = "PHP";
  ?>

  <?php
    /* Here are some imaginary numbers for a possible salary package for the skills we're learning: */
    $base_salary = 158470;
    $bonus = 25815.25;
    $benefits = 0.2;
    //percentage of total
    $time_off = 6476;
    //in dollars
    $benefits = "this is a string now"; // You just changed the value of $benefits from a float number to a string!
  ?>

  <br/>
  <ul>
    <li> My Base Salary might be: <?php echo $base_salary; ?> </li>
    <li> My Bonus might be: <?php echo $bonus; ?> </li>
    <li> My Benefits might be: <?php echo $benefits; ?> </li>
    <li> My Time Off might be worth: <?php echo $time_off; ?> </li>
  </ul>
</ol>
```

Click **Preview**. Those variables should show up now.

1. PHP is part of the LAMP stack.
 - My Base Salary might be: 158470
 - My Bonus might be: 25815.25
 - My Benefits might be: To be determined
 - My Time Off might be worth: 6476



There they are. Actually, instead of displaying themselves, our variables displayed the **values** they were holding.

By the way, they're not just sneaky; they're picky too. Turns out, **variable names** may consist of only **letters, numbers, and the underscore(_) character**. Not just that, the first character of the variable name CANNOT be a number.

Here's a list of valid and invalid variable names:

VALID variable names:

- **`$_var`**
- **`$heres_a_name`**
- **`$t12345`**
- **`$x`**

INVALID variable names:

- **`$1_var`**
- **`$here's-a-name`**

- **\$t+12345**
- **\$x?**

Modifying Variables and Values with Operators

Variables are not useful unless they've been modified. **Operators** can be used to modify variables and their values. Operators are fairly simple to use, in fact, you've already learned one: the **assignment** operator, represented by the equal sign (=). The assignment operator is a quick, easy, and intuitive way to instruct a variable to hold a certain value.

But what if we want to *change* a variable's value?

Type the following into CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is part of the LAMP stack. </li>
  <?php
    $lamp_l = "Linux";
    $lamp_a = "Apache";
    $lamp_m = "MySQL";
    $lamp_p = "PHP";
  ?>
  <?php
    /* Here are some imaginary numbers for a possible salary package for the skills we're learning: */
    $base_salary = 158470;
    $bonus = 25815.25;
    $benefits = 0.2;
    //percentage of total
    $time_off = 6476;
    //in dollars
    $benefits = "this is a string now"; // You just changed the value of $benefits from a float number to a string!
  ?>
  <br/>
  <ul>
    <li> My Base Salary might be: <?php echo $base_salary; ?> </li>
    <li> My Bonus might be: <?php echo $bonus; ?> </li>
    <li> My Benefits might be: <?php echo $benefits; ?> </li>
    <li> My Time Off might be worth: <?php echo $time_off; ?> </li>
    <li> My Base Salary plus Bonus would total: <?php echo $base_salary + $bonus; ?> </li>
  </ul>
</ol>
```

Preview this code. We added the values of the variable's `$base_salary` and `$bonus`. Sweet.

1. PHP is part of the LAMP stack.
 - My Base Salary might be: 158470
 - My Bonus might be: 25815.25
 - My Benefits might be: To be determined
 - My Time Off might be worth: 6476
 - My Base Salary plus Bonus would total: 184285.25

The plus sign (+) is also an **operator**. More specifically, a **binary operator**, since it takes *two* variables or values (in this case, called **arguments**), performs the addition operation on them, and returns the result - just like those shy variables do. In this case, we displayed the result through the "echo" statement.

Below is a list of some binary operators, and some examples of them in action.

Assuming `$i = 12` and `$j = 5` then...

Operator	Name	Usage	Result
----------	------	-------	--------

=	assign	\$i = \$j	5
+	add	\$i + \$j	17
-	subtract	\$i - \$j	7
*	multiply	\$i * \$j	60
/	divide	\$i / \$j	2.4
%	mod (remainder of division)	\$i % \$j	2

Play around with these in your program and see what you get. Seriously, practice! Try applying different operators to the example you've been working on in this lesson, and echo out the results.

The operations (except for addition) need to be executed in the order they appear, from left to right, to work properly.

By the way, the operators above only operate on integers and floating point number values. There are different operators that work on strings. Most specifically, the **concatenation** operator. Here is an example using concatenation. Notice the period in front of \$lamp_l:

Type the following into CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is part of the LAMP stack. </li>
  <?php
    $lamp_l = "Linux";
    $lamp_a = "Apache";
    $lamp_m = "MySQL";
    $lamp_p = "PHP";
    echo "<br />The stack begins with " . $lamp_l;
  ?>
  <?php
    /* Here are some imaginary numbers for a possible salary package for the skills we're learning: */
    $base_salary = 158470;
    $bonus = 25815.25;
    $benefits = 0.2;
    //percentage of total
    $time_off = 6476;
    //in dollars
    $benefits = "this is a string now"; // You just changed the value of $benefits from a float number to a string!
  ?>
  <br/>
  <ul>
    <li> My Base Salary might be: <?php echo $base_salary; ?> </li>
    <li> My Bonus might be: <?php echo $bonus; ?> </li>
    <li> My Benefits might be: <?php echo $benefits; ?> </li>
    <li> My Time Off might be worth: <?php echo $time_off; ?> </li>
    <li> My Base Salary plus Bonus would total: <?php echo $base_salary + $bonus; ?> </li>
  </ul>
</ol>
```

Preview the code and see what happens. After you Preview you should see the following sentence on your page:

The stack begins with Linux.

Let's break it down. How did this:

echo "The stack begins with " . \$lamp_l;

become this?:

The stack begins with Linux.

Well, the first part in quotation marks is a string and the \$lamp_l is a variable holding the string "Linux". To "add" them together, we use a period . which is the concatenation operator. Did you understand that? If not look again...

echo "The stack begins with ".\$lamp_l;

When used properly in PHP, suddenly that lowly punctuation mark, the period (.), becomes a powerful concatenation tool. Yes, the concatenation operator (.) is yet another **binary operator** in PHP, and an extremely useful one at that.

But the most useful characteristic of these operators is that they can be **nested**. To nest operators essentially means we can use them together.

Type the following blue code (notice the periods in red) into CodeRunner:

```
<ol style="font-size:16px;">
  <li> PHP is part of the LAMP stack. </li>
  <?php
    $lamp_l = "Linux";
    $lamp_a = "Apache";
    $lamp_m = "MySQL";
    $lamp_p = "PHP";
    echo "<br />The stack begins with " . $lamp_l . " and goes on to include " . $lamp_
a . ", and " . $lamp_p . "!<br />";
  ?>
  <?php
    /* Here are some imaginary numbers for a possible salary package for the skills we'
re learning: */
    $base_salary = 158470;
    $bonus = 25815.25;
    $benefits = 0.2;
    //percentage of total
    $time_off = 6476;
    //in dollars
    $benefits = "This is a string now."; // You just changed the value of $benefits fro
m a float number to a string!
    $total = $base_salary + $bonus + $time_off;
    $total_compensation = $total + ($total * 0.2); // Adding in benefits
  ?>
  <br/>
  <ul>
    <li> My Base Salary might be: <?php echo $base_salary; ?> </li>
    <li> My Bonus might be: <?php echo $bonus; ?> </li>
    <li> My Benefits might be: <?php echo $benefits; ?> </li>
    <li> My Time Off might be worth: <?php echo $time_off; ?> </li>
    <li> My Base Salary plus Bonus would total: <?php echo $base_salary + $bonus; ?> </
li>
    <li> My total compensation would be <?php echo $total . " without benefits, and " .
$total_compensation . " with benefits."; ?> </li>
  </ul>
</ol>
```

In the code above there is operator nesting happening all over the place. Preview your file. You should get something like this:

1. PHP is part of the LAMP stack.

The stack begins with Linux, and goes on to include Apache, MySQL, and PHP!

- My Base Salary might be: 158470
- My Bonus might be: 25815.25
- My Benefits might be: To be determined
- My Time Off might be worth: 6476
- My Base Salary plus Bonus would total: 184285.25
- Your total compensation would be 190761.25 without benefits, and 228913.5 with benefits.



It appears that operator nesting worked just fine. That's not to say that our **binary** operators started taking on more than two arguments. Instead, we've executed a succession of binary operations, with the one operation taking the results of the last operation into consideration.

Here are a couple more things to consider:

- Why do you suppose the "concat" operator (.) had no problem mixing strings, floats, and integers?
- Were the parentheses (()) necessary in the \$total_compensation line?
- What role do you think the parentheses play?

Finally, here are some useful PHP "shortcut" operators. These operators reduce the need for nesting to execute some common tasks and they are really handy. Can you figure out which operators are **unary operators**? (Hint: unary operators need only *one* argument.)

Play around with these and see what you get:

Operator	Equivalent
<code>\$i += \$j</code>	<code>\$i = \$i + \$j</code>
<code>\$i -= \$j</code>	<code>\$i = \$i - \$j</code>
<code>\$i *= \$j</code>	<code>\$i = \$i * \$j</code>
<code>\$i /= \$j</code>	<code>\$i = \$i / \$j</code>
<code>\$i++</code>	<code>\$i = \$i + 1</code>
<code>\$i--</code>	<code>\$i = \$i - 1</code>
<code>\$i .= \$j</code>	<code>\$i = \$i . \$j</code>

Superglobals

PHP has a set of predefined variables to make our lives easier. Superglobals can be accessed by classes, functions, or files at any time without having to do anything special! Very nice. So, what are these *Superglobals*?

Before we delve too deeply, let's get a small taste of what's to come. After all, we can't simply give out all the secrets in the beginning. There wouldn't be anything to look forward to!

\$GLOBALS

1. References all variables that are in the global scope.
2. Associative array.
3. Variable names are keys of \$GLOBALS array.

CODE TO TYPE: \$GLOBALS example

```
<?php
function testScope() {
    echo "The variable in the main code doesn't extend to within the function: $scope<br>";

    //assign a value to the variable named $scope that IS within function scope
    $scope = "WITHIN FUNCTION";
    echo "The local scope within the function: $scope<br>";

    //the superglobal DOES extend within the function
    echo "The global scope: {$GLOBALS['foo']}<br>";
}

//define $scope in the main code
$scope = "MAIN CODE";
echo "The local scope in the main code body: $scope<br>";

//define a global value (
$GLOBALS['foo'] = "SUPERGLOBAL";
echo "The value in the superglobal is {$GLOBALS['foo']}<br>";

//now run function, which has separate scope and $scope variable
testScope();

//show that main code's $scope is unaffected
echo "The local scope in the main code body: $scope<br>";
?>
```

The local scope in the main code body: MAIN CODE

The value in the superglobal is SUPERGLOBAL

The variable in the main code doesn't extend to within the function:

The local scope within the function: WITHIN FUNCTION

The global scope: SUPERGLOBAL

The local scope in the main code body: MAIN CODE

\$_SERVER:

1. Array containing information to Headers, Paths, and Script locations.
2. Entries generated by web server.

CODE TO TYPE: \$_SERVER example

```
<?php
echo $_SERVER['SERVER_NAME'];
?>
```

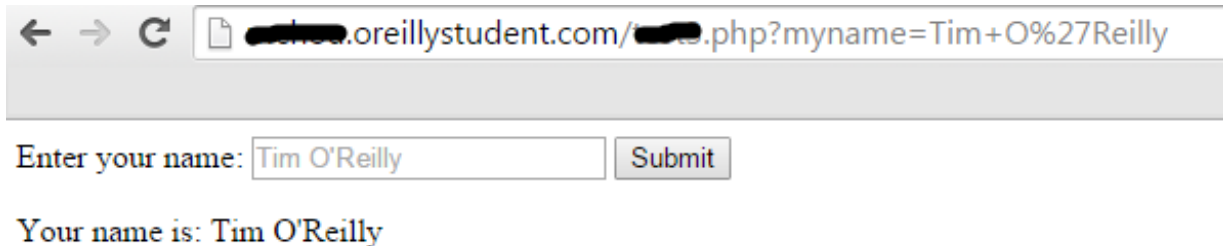
mchou.oreillystudent.com

\$_GET

1. Associative array.
2. Populated by URL parameters.

CODE TO TYPE: \$_GET example

```
<form action="" method="get">
Enter your name: <input type="text" name="myname" placeholder="Tim O'Reilly"/>
<input type="submit" />
</form>
<?php
    echo "Your name is: " . htmlspecialchars($_GET["myname"]);
?>
```



Enter your name:

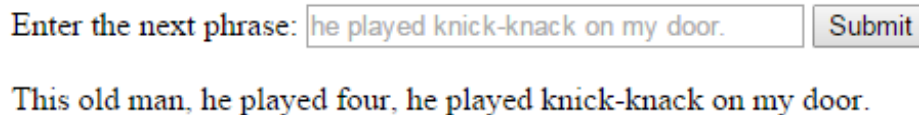
Your name is: Tim O'Reilly

\$_POST

1. Associative array.
2. Array is passed via HTTP POST method.

CODE TO TYPE: \$_POST example

```
<form action="" method="post">
Enter the next phrase: <input type="text" name="next_phrase" size="50" placeholder="he played knick-knack on my door."/>
<input type="submit" />
</form>
<?php
    echo "This old man, he played four, " . htmlspecialchars($_POST["next_phrase"]);
;
?>
```



Enter the next phrase:

This old man, he played four, he played knick-knack on my door.

Note

This is not an exhaustive list of PHP's Superglobals; however, click [here](#) for a full list with examples, definitions, and a peek of what's to come!

Phew! We've covered a lot of ground. Don't forget to **Save** your work, and hand in the assignments from your syllabus. See you at the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Decisions

In the last lesson we learned that storing values in variables and manipulating them with operators are among the most important tools we have for programming in PHP. Now let's talk about automating repetitive tasks and the decisions you'll make based on the values present in your programs.

Comparison Operators and Conditions

We make comparisons everyday. When we shop, we look at prices of similar items to determine which deals are best. When we take a trip, we compare alternative routes to decide which will be most expedient. Well, it turns out that we can do comparisons in PHP and other computer languages as well. Let's look at this process using a simple example. Let's try a comparison of **Captain Crunch** breakfast cereal to **Frosted Flakes** breakfast cereal.

Suppose Captain Crunch is 4 dollars a box, while Frosted Flakes is 5 dollars. We can use PHP to figure out which price is greater. (We realize you can determine this fairly easily without PHP, but let's go ahead and work the example anyway so we can see how PHP works.)

Add the following BLUE and GREEN code to your file in CodeRunner:

```
<?php
    $captain_crunch = 4;
    $frosted_flakes = 5;
?>
Does Captain Crunch cost less than Frosted Flakes? <?php echo ($captain_crunch < $frosted_flakes); ?>
<br />
<br />
Or, is Captain Crunch priced greater than Frosted Flakes? <?php echo ($captain_crunch > $frosted_flakes); ?>
```

Preview that. What was returned from the echo command here?

In the the last lesson we learned to modify variables using some of the standard operators: add, subtract, concatenate, and others. The **comparison operators** we're using now compare two variables, producing TRUE or FALSE results.

For instance, the program above compares two integers to determine if one is larger than the other. (" $<$ " is a symbol that means "less than," while " $>$ " means "greater than").

This statement is TRUE:

4 < 5

We all know that 4 is less than 5.

This statement is FALSE:

4 > 5

So how did your program answer the question it was asked about Captain Crunch and Frosted Flakes?

Here's what you saw:

Does Captain Crunch cost less than Frosted Flakes? 1

Or, is Captain Crunch priced greater than Frosted Flakes?



Obviously, in our example, Captain Crunch is less (expensive) than Frosted Flakes, but what's with the number one (1) at the end there? Was that a typo?

No, that wasn't a typo. Turns out, this is how PHP interprets the **Boolean** result "TRUE." (Boolean, by the way, is just a fancy programming word referring to the results of "true or false" inquiries.) Similarly, instead of returning "FALSE" or "no" when asked if Captain Crunch is greater than Frosted Flakes, your program returned the **NULL** character. "Null" is computer-speak for "nothing." When things are false, nothing gets returned, so nothing is printed.

Here's a table of values that PHP can interpret as TRUE or FALSE:

FALSE	TRUE	Notes
0 (zero)	any non-zero number	non-zero examples: 1, -1, 0.5
false	true	no quotes ("), otherwise it's just a string
NULL, null, "", or ""	any non-null string	The space (" ") character is NOT the same as the null ("") character

Here's a list of comparison operators you can experiment with in your program:

Operator	Name	Usage	Result
==	Equal	<code>\$a == \$b</code>	TRUE if \$a and \$b are equal.
===	Identical	<code>\$a === \$b</code>	TRUE if \$a and \$b are equal AND if they are of the same type (ie \$a and \$b are both integers).
!= <>	Not equal	<code>\$a != \$b</code> <code>\$a <> \$b</code>	TRUE if \$a and \$b are not equal.
!==	Not identical	<code>\$a !== \$b</code>	TRUE if \$a is not equal to \$b OR if \$a and \$b are not of the same type.
<	Less than	<code>\$a < \$b</code>	TRUE if \$a is less than \$b.
>	Greater than	<code>\$a > \$b</code>	TRUE if \$a is greater than \$b.
<=	Less than or equal to	<code>\$a <= \$b</code>	TRUE if \$a is less than OR equal to \$b.
>=	Greater than or equal to	<code>\$a >= \$b</code>	TRUE if \$a is greater than OR equal to \$b.

Note

In PHP we use two equal signs (==) to test for equality. (When you use "===" you're essentially asking: **Are these values equal?**). Two equal signs, like `$a == $b`, compare \$a to \$b (in English it would read "is \$a equal to \$b?"), whereas one equal sign `$a = $b` assigns \$b to \$a (in English it would read "set \$a is equal to \$b").

IF and ELSE Control Structure

You may not know it, but you actually already understand **if** and **else** control structures. You use them everyday when you decide things like **"I'll buy Captain Crunch if it's less expensive than Frosted Flakes, or else I'll buy Frosted Flakes"**. You've set conditions and also decided on an alternative course of action should those conditions fail to be met. In a program, this is called a control structure.

In PHP, you would write the sentence above like this:

Type the following into a new file in CodeRunner:

```
<?php $Captain_Crunch = 4; $Frosted_Flakes = 5; if ($Captain_Crunch
    < $Frosted_Flakes) { echo "I'll buy Captain Crunch"; }
    else { echo "I'll buy Frosted Flakes"; } ?>
```

Preview the code above. Which cereal does PHP instruct you to buy? Try changing the numbers assigned to `$Captain_Crunch` and `$Frosted_Flakes` to see what happens.

if statements have a specific form.

OBSERVE:

```
if (expression) {
    statement(s) executed if expression is TRUE } else {
    (optional) statement(s) executed if expression IS FALSE
}
```

Again, this **if** statement is also referred to as a **control statement**. PHP first evaluates the **expression** to see if it is true or false. If the **expression** is true, then the statements in **blue** are executed. If not, then the statements in **BLUE** are not executed, but the **green** ones are.

Logical Operators

Questions can be more complicated than statements. For instance, if Captain Crunch is more (expensive) than Frosted Flakes, but Fruit Loops are less (expensive) than Frosted Flakes, we might want to choose Fruit Loops. The following code can handle these kinds of complications:

Add the colored code below into your document in CodeRunner:

```
<?php $Captain_Crunch = 5; $Frosted_Flakes = 4;
    $Fruit_Loops = 3;
    if ($Captain_Crunch < $Frosted_Flakes) {
        echo "I'll buy Captain Crunch";
    } else if ($Captain_Crunch > $Frosted_Flakes && $Frosted_Flakes > $Fruit_Loops) {
        echo "I'll buy Fruit Loops.";
    } else { echo "I'll buy Frosted Flakes.";
    }
?>
```

Preview this code. Which cereal does PHP recommend that you buy? Try changing the numbers representing the prices and observe the results.

In this example, we've added a couple of things for your consideration. First we added a **logical operator**. We used **&&** which simply means **AND**. The other addition was the **else if**. We can have as many of those within an **if** statement as we need.

So now the line reads:

OBSERVE:

```
else if ($Captain_Crunch > $Frosted_Flakes && $Frosted_Flakes > $Fruit_Loops) {
    echo "I'll buy Fruit Loops.";
```

In English, the line reads:

OBSERVE:

```
"Or else if Captain Crunch is greater than Frosted Flakes,
AND
Frosted Flakes is greater than Froot Loops, then I'll buy Fruit Loops.
```

Notice that when Captain Crunch is 6 dollars **and** Frosted Flakes is 5 dollars **and** Fruit Loops is 4 dollars, **then** **\$Captain_Crunch > \$Frosted_Flakes** is **TRUE**, and that **\$Frosted_Flakes > \$Fruit_Loops** is **TRUE**. So the whole thing is **TRUE** and so you'll buy Fruit Loops!

Here are some rules to remember about logical operators:

- (**TRUE AND TRUE**) is **TRUE**
- (**TRUE AND FALSE**) is **FALSE**
- (**TRUE OR FALSE**) is **TRUE**
- (**FALSE OR FALSE**) is **FALSE**

Like comparison operators, the logical operator performs a comparison on two arguments, and returns a **TRUE** or **FALSE** (1 or null) answer. However, the logical operator compares things that are already **TRUE** or **FALSE**.

Below is a list of logical operators.

Operator	Name	Usage	Result
AND	AND	\$a AND \$b	TRUE if \$a and \$b are TRUE

&&	AND	\$a && \$b	TRUE if \$a and \$b are TRUE.
OR 	OR	\$a OR \$b \$a \$b	TRUE if \$a or \$b is TRUE.
XOR	Exclusive OR	\$a XOR \$b	TRUE if \$a OR \$b is TRUE, but not both.

Look again at this condition:

```
($Captain_Crunch >= 3 && $Frosted_Flakes < 10)
```

Remember that nested operators perform in a certain order, depending on certain rules? There are rules here too. Specifically, the comparison operators are evaluated *before* the logical evaluator. This way, the logical evaluator only needs to look at the TRUE or FALSE results, and act accordingly.

Like this:

```
((($Captain_Crunch >= 3) &&
($Frosted_Flakes < 10)) ( (TRUE) &&
(TRUE)) (TRUE))
```

Operator nesting is really useful. And fortunately, you can do it with logical operators too.

Type the following into CodeRunner:

```
<?php
    $Captain_Crunch = 5;
    $Frosted_Flakes = 4;
    $Fruit_Loops = 5;
    $Oatmeal = 2;
    if ($Captain_Crunch < $Frosted_Flakes) {
        echo "I'll buy Captain Crunch";
    } else if ($Captain_Crunch > $Frosted_Flakes && $Frosted_Flakes > $Fruit_Loops) {
        echo "I'll buy Fruit Loops.";
    } else if ($Captain_Crunch > 4 && $Fruit_Loops > 4 && $Oatmeal < 4 ) {
        echo "I'll get some Oatmeal.";
    }
?>
```

Play with this one for a while—try entering different values for the different cereals. Be sure to Preview often to see what happens.

NOTICE: Before you move on, Save the PHP file you've been working on as **compare.php**.

A Brief Preview of Forms

Before we continue on with control structures, let's make these examples a little more interesting by getting your PHP program to take input from a user on the web. We're going to make a form that will help us understand these control structures better. This is a brief introduction. We'll cover forms in much more detail later in the course.

So far in this lesson, we've been changing the values in the variables **\$Captain_Crunch**, **\$Frosted_Flakes**, **\$Fruit_Loops**, and **\$Oatmeal** by hand. Typically though, control structures evaluate changes made to variables and then react to those changes. If the user changes an input, we can account for all the possibilities through control structures.

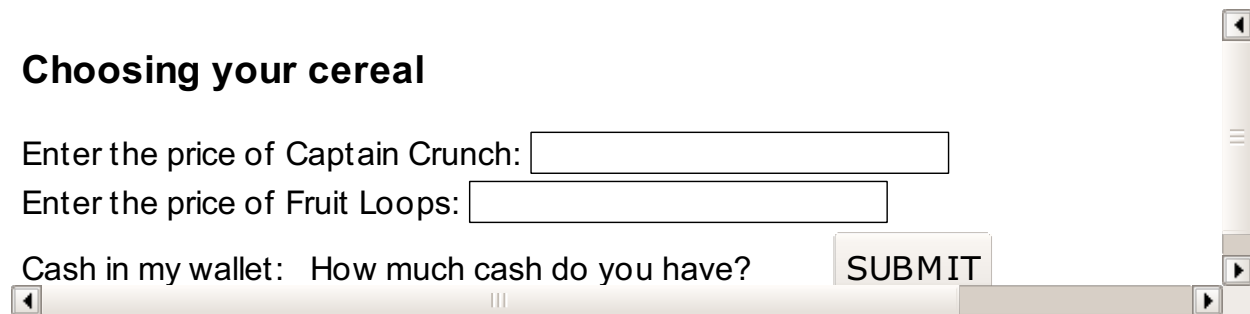
To make our program more interactive, we're going to make a web page with a few input forms we'll use to submit values to our PHP program. Then we're going to take those inputs and assign them to the variables listed above.

Let's make an HTML form.

Make sure you're in HTML and type the following into CodeRunner:

```
<body>
<h3>Choosing your Cereal</h3>
<form method="GET" action="compare.php"> Enter the price of Captain Crunch:
  <input type="text" size="25" name="crunch_price" value="" />
<br /> Enter the price of Fruit Loops: <input type="text" size="25"
  name="loops_price" value="" />
<br /> Cash in my wallet: <select name="cash_money">
<option value="">How much cash?</option>
<option value="1">$1.00</option>
<option value="2">$2.00</option>
<option value="3">$3.00</option>
<option value="4">$4.00</option>
<option value="5">$5.00</option>
<option value="10">$10.00</option>
</select>
<input type="submit" value="SUBMIT" />
</form>
</body>
```

Now Preview this in HTML. If you select an item and click submit, it won't do anything. You should see a page that looks like this:



Save this page as **userinput.html**, or anything you like, so long as you can remember the name.

Now we've made a web page that will take input from a web user, and then send that input to the PHP program that we'll use to process it. Now we just need to make our PHP program retrieve the input. To do this, we have to use something called a **superglobal array**. Now that's a mouthful! We'll actually study superglobals in detail in a later lesson. For now let's just try it!

Switch back to **PHP** with the **compare.php** PHP program we've been using, and make the following changes.

Type the changes in BLUE into CodeRunner:

```
<?php
$Captain_Crunch = $_GET["crunch_price"];
$Frosted_Flakes = 4;
$Fruit_Loops = $_GET["loops_price"];
$Oatmeal = 2;
$my_cash = $_GET["cash_money"];
$total = $Captain_Crunch + $Frosted_Flakes;
if ($total < $my_cash) {
    echo "I'll buy both Captain Crunch and Frosted Flakes!";
} else if ($Captain_Crunch < $my_cash) {
    echo "I'll buy Captain Crunch.";
} else if ($Captain_Crunch > $my_cash && $Fruit_Loops < $my_cash) {
    echo "I'll buy some Fruit Loops.";
} else {
    echo "Forget it, I'm going home.";
}
?>
```

Now Save this PHP program as **compare.php**, then go back to your **userinput.html** file in HTML. Preview

it. Enter different prices for the two cereals, select the amount of cash you have in your wallet, then click submit. Now your program should change according to the input you submitted in the form. Cool, huh?

We're just getting started with **control structures**, so be sure to save your work and hand in your assignments. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Multiple Control Structures and Loops

Let's continue on from the last lesson. Make sure you've opened **userinput.html** in **HTML**, and **compare.php** in **PHP**. Ready? Let's go!

Multiple Control Structures

In the last lesson we introduced the **else if** statement, which allows us to work with multiple conditions.

Check out this else if statement:

```
<?
    if ($total < $my_cash) {
        echo "I'll buy both Captain Crunch and Frosted Flakes!";
    }
    else if ($Captain_Crunch < $my_cash ) {
        echo "I'll buy Captain Crunch.";
    }
    else if ($Captain_Crunch > $my_cash && $Fruit_Loops < $my_cash ){
        echo "I'll get some Fruit Loops.";
    }
    else {
        echo "Forget it, I'm going home.";
    }

?>
```

When the first **if** statement fails, PHP checks the **else if** statement before going on to **else**. As long as you begin with an **if** and end with an **else**, (if you have a default action), you can add any number of **else if** statements in the middle. This way we can check and react to a lot of conditions. Let's add some conditions to the example we worked on last lesson.

Type the code in blue into your PHP program in CodeRunner:

```
<?

$Captain_Crunch = 5;
$Frosted_Flakes = 4;
$Fruit_Loops = 3;
$Oatmeal = 2;
$my_cash = $_GET["cash_money"];

$total = $Captain_Crunch + $Frosted_Flakes;

if ($my_cash == 10) {
    echo "I'll buy both Captain Crunch and Frosted Flakes!";
}
else if ($my_cash == 5 ) {
    echo "I'll buy Captain Crunch.";
}
else if ($my_cash == 4){
    echo "I'll buy Frosted Flakes.";
}
else if ($my_cash == 3 ){
    echo "I'll buy Fruit Loops.";
}
else if ($my_cash == 2){
    echo "I'll buy Oatmeal.";
}
else {
    echo "Forget it, I'm going home.";
}

?>
```

Save this code using the filename `compare.php`, and open your **userinput.html** file in HTML. Try entering some numbers for the amount of money you have and Preview it (the other variables are set in the program, so the input for those fields won't matter in this example).

Even though the above code works just fine, the procedure could be streamlined by using a **switch** statement. The `switch` control structure is similar to the `if` control structure, but it's especially useful when you have one variable with many possible values. The `switch` control structure is a more efficient means of accomplishing the same task. It's up to you decide which control structure you like better.

This is how we'd change our code into a switch statement:

```
switch($my_cash) {
case "10":
echo "I'll buy both Captain Crunch and Frosted Flakes.";
break;
case "5":
echo "I'll buy Captain Crunch.";
break;
case "4":
echo "I'll buy Frosted Flakes.";
break;
case "3":
echo "I'll buy Fruit Loops.";
break;
case "2":
echo "I'll buy Oatmeal.";
break;
default:
echo "Forget it, I'm going home.";
}
```

Give it a try. Save the old file as new_file.php (don't forget to make a new HTML file as well), then replace the block of code that contains the **if** statements with the switch statements above. Use whichever method you prefer, it's your call.

And of course we want you to practice! Especially since we're going to assign this task as an objective later.

Before we go on, experiment and find answers to these questions:

- What happens when you **nest** comparison operators?
- Would (null == 0) be TRUE or FALSE?
- How about (null === 0)?
- In an **if** statement, do you have to have parentheses **()** around the condition?
- What about brackets **{ }**?
Hint: Try this with one action statement AND with two.
- Can you put the *whole* **if** control structure in one line?
- Do you always need to have an **else** statement?
- When nesting **logical** operators, do you need parentheses?
- What happens when you remove **break;** statements?

WHILE and FOR Loops

A **loop** is a repetitive task that goes on *while* something is true or *for* some number of steps. That's why they are called "while" and "for" loops.

A **while loop** has the following structure:

```
while (something is true) { do some stuff };
```

As soon as that something is false, the while loop stops.

Whereas a **for loop** has the following structure:

```
for (some number of steps) { do some stuff };
```

Let's look at an example. Type this into a new PHP file in CodeRunner:

```
<?

    echo "Hide and go seek, I'm counting to 25:<br>";

    $counts = 1;
    while ($counts <= 25) {
        echo $counts." Mississippi...<br>";
        $counts++;
    }

    echo "Ready or not, here I come!<br>";

?>
```

In case you didn't play hide-and-seek in your childhood, this is how you'd count out loud while giving the other kids a chance to hide. Such fun!

We have introduced a new form of control structure - the **WHILE loop**. And like with any control structure, the WHILE loop does something in response to a TRUE conditional statement. In this case, however, the loop continues to

repeat the action *until* the conditional statement is FALSE.

All loops have four essential parts:

1. The **initial value statement**, in this case `$counts = 1;`
2. The **conditional statement**, in this case `$counts <= 25`
3. The **action statement(s)**, in this case `echo $counts." Mississippi...
;`
4. The **increment statement**, in this case `$counts++;` (Remember this **unary operator**?)

In order for a loop to work, it has to have a starting point, an ending point, and something to do in between. What would happen if any of the four elements in our example were missing? Try messing with them, and you'll find out. (Go ahead, try. I'll wait.)

Note

Loops follow the same scheme as any control structure, in that you can **nest** all kinds of conditional statements and actions within them, including more loops. This can be lots of fun -- especially for duping your buddies into thinking the computer screen is possessed by gremlins!

Our *counting* loop example above is a really common loop--so common, in fact, that almost all programming languages have developed an alternate type of loop that can be used as a shortcut: the **FOR loop**.

The FOR loop structure looks like this:

```
for ($counts = 1; $counts <= 25; $counts++) {  
    echo $counts." Mississippi...<br>;  
}
```

Try replacing your WHILE loop with this FOR loop, then Preview the code. You should see the exact same result. In fact, the FOR loop has exactly the same four elements as the WHILE loop. The only difference is the order of the elements in the syntax. Well, that, and if you forget the **increment statement** in this one, PHP will yell at you. Sounds mean, but sometimes we need a little kick to keep from inadvertently causing an **infinite loop**. Yuck.

You may think that learning both WHILE loops and FOR loops in PHP is needless and redundant, and it's true that most of the time they are interchangeable. However, as your scripts gain more complexity, you'll find that some tasks are a perfect fit for using **FOR**, while using **WHILE** is best for others.

You've come really far! Now you can program the majority of what you'll need in PHP. Congratulations! You are one of the best PHP programmers in the world!

...Circa 1995, that is. To create cutting-edge web software for *this* century, we have a ways to go. Take heart—you've accomplished a lot already. Save your work, and don't forget the assignments. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Functions

A **function** acts like a small program within your larger program. You invoke a function, send it information, and get something back. A great way to learn to use functions, is to create one and use it. Let's go!

Creating Code Reusability with Functions

Let's create a new program so we can practice using functions. Our new program will take user input just like our previous programs, only this time we'll inquire about the user's state of mind and recommend a mantra for them to repeat.

Make sure you're in HTML syntax and type the following into CodeRunner:

```
<body>

  <h3>OST's Mantra generator</h3>

  <form method="GET" action="mantra.php">

    My current mood:
    <select name="my_mood">
      <option value="">Please choose...</option>
      <option value="happy">I'm happy.</option>
      <option value="sad">I'm sad.</option>
      <option value="angry">I'm angry.</option>
      <option value="indifferent">I'm indifferent.</option>
    </select>

    <input type="submit" value="SUBMIT" />

  </form>

</body>
```

Save this HTML file as **moodinput.html**

In PHP Mode, type the following code highlighted in BLUE into CodeRunner:

```
<?
$my_mood = $_GET["my_mood"];

if ($my_mood == "happy") {

    echo "Repeat the following: <br><br>";

    for ($chant = 1; $chant <= 10; $chant++) {
        echo " OM... ";
    }

}
else if ($my_mood == "sad") {

    echo "Repeat the following: <br><br>";

    for ($chant = 1; $chant <= 10; $chant++) {
        echo " okay... ";
    }

}
else if ($my_mood == "angry") {

    echo "Repeat the following: <br><br>";

    for ($chant = 1; $chant <= 10; $chant++) {
        echo " Mississippi... ";
    }

}
else if ($my_mood == "indifferent") {

    echo "Repeat the following: <br><br>";

    for ($chant = 1; $chant <= 10; $chant++) {
        echo " Wake up... ";
    }

}
else {

    echo "Repeat the following: <br><br>";

    for ($chant = 1; $chant <= 10; $chant++) {
        echo " Try harder... ";
    }

}

?>
```

Save this code as **mantra.php** and, once again, open up the HTML file moodinput.html.

Preview the HTML. Enter different values for your mood (after all, we're all pretty moody).

We have used pretty much the same PHP code in several places. When you have the same code or similar code in lots of different places, use a **function**. It will make your program more readable and save time. Say we decided to recommend chanting our mantra 20 times instead of 10. Unless we used a function, we'd have to change the code by hand in each of the "for" loops. Even for a simple code like ours, that would be truly annoying. You can imagine trying to do this with long and complicated code--it could get downright ugly. Fortunately, **functions** enable us to avoid such unpleasantness. We can create a function to execute a particular task each time we enter the name of the function into our program. This is referred to as "calling" a function. Change your program so it looks like the stuff below—be sure to remove the "for" loops within the "else if" statements.

Switch back to PHP and add the following code in BLUE into CodeRunner:

```
<?
function mantra($the_sound) {
    for ($chant = 1; $chant <= 10; $chant++) {
        echo $the_sound . "... ";
    }
}

$my_mood = $_GET["my_mood"];
if ($my_mood == "happy") {
    mantra("OM");
}
else if ($my_mood == "sad") {
    mantra("okay");
}
else if ($my_mood == "angry") {
    mantra("mississippi");
}
else if ($my_mood == "indifferent") {
    mantra("Wake up");
}
else {
    mantra("Try harder");
}
?>
```

Save this file as **mantra.php** and then switch back to your **moodinput.html** file. Try entering different moods. You should get the same results as before, but this time you used functions.

Each time the **mantra("something")** function is encountered by PHP, it calls the function definition **function mantra(\$the_sound)**. The variable (**\$the_sound** in this case) is set to whatever is in between the parentheses when **mantra("something")** is encountered. For instance, **mantra("OM")** is calling the **function mantra(\$the_sound)** and setting **\$the_sound = "OM"**. This is known in functions as "setting a parameter." This particular function takes one parameter: **\$the_sound**. However, functions can take no parameters at all or many different parameters.

Much like a variable holds values, **functions** hold **processes** (snippets of code) that we want to reuse. So instead of having to add the same code over and over again, we can simply *call* the **function**. In this case, when **mantra()** is encountered, the code inside of the brackets { and } in the definition **function mantra(){ }** is executed. Functions will only be executed when they are *called*. Try removing the calls to **mantra()**. You'll see that the function doesn't do anything then.

Note

If the function you've created doesn't have any parameters, you still need to have the parentheses in place, they just won't contain anything. Notice how no dollar signs (\$) are used in the function name **mantra**, but instead we follow the name with parentheses(**()**). They need to be there, that's just the way it is.

Congratulations—you've just worked through your first example of **code reusability**!

Function and Variable Scopes

Scope refers to a variable's area of influence. If a variable is defined inside of a function, then its area of influence is only within that function. That means we can use that variable name again in another function--setting values to it in one function won't affect the setting in another function. Let's try using a function to encapsulate those "if" statements in our example from the last section. In the process, we can see how scope may affect the outcome of our program.

Revise your PHP program so it looks like this in CodeRunner:

```
<?
function mantra($the_sound) {
    for ($chant = 1; $chant <= 10; $chant++) {
        echo $the_sound . "... ";
    }
}

function Mood_Chant(){

    if ($my_mood == "happy") {
        mantra("OM");
    }
    else if ($my_mood == "sad") {
        mantra("okay");
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
    }
    else if ($my_mood == "indifferent") {
        mantra("Wake up");
    }
    else {
        mantra("Try harder");
    }
}

$my_mood = $_GET["my_mood"];

Mood_Chant();

?>
```

Save this as **mantra.php**, then open your **moodinput.html** file. Try altering ANY of the moods on the list. No matter what you choose, this code will always return **Try Harder** as the output.

So why is the program returning **Try Harder** as a result, no matter what we select? Let's perform some diagnostic tests to find out. We'll enter some echo statements to print out variable values in different parts of our program. Then we can use the information we get to determine the path our program is taking and the steps we need to take to correct our problem. Let's try it.

Add some echo statements into CodeRunner:

```
<?
function mantra($the_sound) {

    for ($chant = 1; $chant <= 10; $chant++) {
        echo $the_sound . "... ";
    }

}

function Mood_Chant(){

echo "INSIDE the Mood_Chant function, your mood is ".$my_mood."<br>";

    if ($my_mood == "happy") {

        mantra("OM");

    }
    else if ($my_mood == "sad") {

        mantra("okay");

    }
    else if ($my_mood == "angry") {

        mantra("mississippi");

    }
    else if ($my_mood == "indifferent") {

        mantra("Wake up");

    }
    else {

        mantra("Try harder");

    }

}

$my_mood = $_GET["my_mood"];

echo "OUTSIDE the Mood_Chant function, your mood is ".$my_mood."<br>";

Mood_Chant();

?>
```

Once again, Save this as **mantra.php**, then go back to your **moodinput.html** page. Select **angry** from the drop down list and submit it.

You should get something like this:

OUTSIDE the Mood_Chant function, your mood is angry.

INSIDE the Mood_Chant function, your mood is .

Try Harder...Try Harder...Try Harder...Try Harder...Try Harder...Try Harder...Try Harder...Try Harder...Try Harder...



Look closely. What printed out? What didn't? The first result printed out **OUTSIDE the Mood_Chant function, your mood is angry**. We asked PHP to print it with the statement **echo "OUTSIDE the Mood_Chant function, your mood is ".\$my_mood."
";**. So as expected, the variable `$my_mood` was defined as *angry*. However, the second result printed **INSIDE the Mood_Chant function, your mood is .** Even though we asked PHP to print **echo "INSIDE the Mood_Chant function, your mood is ".\$my_mood."
";**, it didn't print a value for `$my_mood`.

In the above example, you would think the value of `$my_mood` ("angry") would print both inside and outside of the function `Mood_Chant()`. But, once the function was called, the value `$my_mood` wasn't seen INSIDE of the `Mood_Chant()` function at all. This is because the variable `$my_mood` is completely different depending on whether it is located outside or inside of the function. Although variables may share the same name, their location determines their effect on the program. When a variable within a function is *encapsulated*, as if the function was its own program, this is referred to in programming as the function's **scope**.

In the next section, we'll learn to set parameters so that scope doesn't prevent us from using functions to the fullest.

Note

PHP isn't as strict with scope as some other languages are. Since PHP isn't strongly typed, you're not required to **declare** variables before you use them. Therefore, within a PHP function, a variable declared within a loop will retain its value outside of that loop. To see this concept at work, try using `echo` to output `$chant` after the **for loop** is finished in `mantra()`.

Using Functions with Parameters and Return Values

As interesting as scope can be, it doesn't help lighten your work load. What's the use of reusing your code in a function, if you have to re-define `$my_mood` within the function? Worse, what if you want to have different values for `$my_mood` anytime you use the function `Mood_Chant()`? We could save ourselves a lot of work if we could feed our function different values and get an output each time. We already did this in the first section above using **parameters**.

Sneaking In with Parameters

Type the following into CodeRunner:

```
<?
function mantra($the_sound) {
    for ($chant = 1; $chant <= 10; $chant++) {
        echo $the_sound . "... ";
    }
}

function Mood_Chant($my_mood){
echo "INSIDE the Mood_Chant function, your mood is ".$my_mood."<br>";

    if ($my_mood == "happy") {
        mantra("OM");
    }
    else if ($my_mood == "sad") {
        mantra("okay");
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
    }
    else if ($my_mood == "indifferent") {
        mantra("Wake up");
    }
    else {
        mantra("Try harder");
    }
}

$my_mood = $_GET["my_mood"];

echo "OUTSIDE the Mood_Chant function, your mood is ".$my_mood."<br>";
Mood_Chant($my_mood);

?>
```

Save this as **mantra.php**, open **moodinput.html**, and select *angry* from the drop-down list. This time, you should have gotten the results you expected.

Look at your function again:

```
function Mood_Chant($my_mood) {  
  
    //code that processes the value of $my_mood  
  
}  
  
.  
.  
.  
  
//passing the value of $my_mood UP to the Mood_Chant function above  
Mood_Chant($my_mood);
```

Passing a **parameter** essentially drills through the wall of your function's scope, making it a more useful machine.

Whatever **parameter** we call to **Mood_Chant(parameter)**; becomes the value for **\$my_mood**. And you don't even have to use the name **\$my_mood**, since it's a completely different variable within the function and outside the function. Try using this on your own.

Look at your function again:

```
function Mood_Chant($my_mood) {  
  
    //code that processes the value of $my_mood  
  
}  
  
.  
.  
.  
  
//passing the value of $my_mood up to the Mood_Chant function above  
Mood_Chant("happy");
```

The value of **\$my_mood** inside of the function **Mood_Chant(\$my_mood)** is **"happy"**. It's like setting **\$my_mood = "happy"** INSIDE of the function.

Now that we've *snuck in* with parameters, let's *sneak out* with return values.

Sneaking out with Return Values

In the examples above, we saw that we can sneak into a function using parameters. We can also sneak out using **return** values. The best way to understand "return" is to use it. Let's get to it.

Type the following into CodeRunner:

```
<?
function mantra($the_sound) {
    for ($chant = 1; $chant <= 10; $chant++) {
        echo $the_sound . "... ";
    }
}

function Mood_Chant($my_mood){
    if ($my_mood == "happy") {
        mantra("OM");
        $after_chant = "<br>I feel serene now.";
    }
    else if ($my_mood == "sad") {
        mantra("okay");
        $after_chant = "<br>I feel better now.";
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
        $after_chant = "<br>I've calmed down now.";
    }
    else if ($my_mood == "indifferent") {
        mantra("Wake up");
        $after_chant = "<br>I'm awake now.";
    }
    else {
        mantra("Try harder");
        $after_chant = "<br>I'll try harder now.";
    }

    return $after_chant;
}

$my_mood = $_GET["my_mood"];

$after_chant_mood = Mood_Chant($my_mood);

echo $after_chant_mood;

?>
```

Save this as **mantra.php**, open up **moodinput.html**, and select anything you like from the drop down list. Now you should get the chant and at the end you should have an "after-chant mood" expressed. All we did here was add some statements into the variable `$after_chant` and then use **return \$afterchant** at the end of the function. When we use return, we are setting a value in place of the function.

But instead of just letting a parameter sneak in, you've allowed a **return value** to sneak *out* of the function scope. Suddenly, your function is an efficient factory, taking in raw ingredients (parameters) and spitting out a refined product -- that is, it *returned a value*. Allowing a return value to sneak out of the function scope is used often in programming to return true or false values in functions that perform tests.

Multiple Parameters and Default Values

We practiced using parameters earlier in the lesson and now we can pass parameters to a function. Let's change our function so that the end user can set how many times we chant our mantra.

Type the following code into your moodinput.html file in CodeRunner:

```
<body>

    <h3>OST's Mantra generator</h3>

    <form method="GET" action="mantra.php">

        My current mood:
        <select name="my_mood">
            <option value="">Please choose...</option>
            <option value="happy">I'm happy.</option>
            <option value="sad">I'm sad.</option>
            <option value="angry">I'm angry.</option>
            <option value="indifferent">I'm indifferent.</option>
        </select>

        Pick a number:
        <select name="my_number">
            <option value="2">Please choose...</option>
            <option value="10">10</option>
            <option value="20">20</option>
            <option value="30">30</option>
            <option value="40">40</option>
        </select>

        <input type="submit" value="SUBMIT" />

    </form>
</body>
```

Save this as **moodinput.html** again. We've added the option of selecting a number, so let's change our program to accept this information and process it.

Type the following in your PHP file in CodeRunner:

```
<?

function mantra($the_sound,$the_number = 10) {

    for ($chant = 1; $chant <= $the_number; $chant++) {
        echo $the_sound . "... ";
    }

}

function Mood_Chant($my_mood, $chant_number = 10){

    if ($my_mood == "happy") {

        mantra("OM",$chant_number);
        $after_chant = "<br>I feel serene now.";

    }
    else if ($my_mood == "sad") {

        mantra("okay",$chant_number);
        $after_chant = "<br>I feel better now.";

    }
    else if ($my_mood == "angry") {

        mantra("mississippi",$chant_number);
        $after_chant = "<br>I've calmed down now.";

    }
    else if ($my_mood == "indifferent") {

        mantra("Wake up",$chant_number);
        $after_chant = "<br>I'm awake now.";

    }
    else {

        mantra("Try harder",$chant_number);
        $after_chant = "<br>I'll try harder now.";

    }

    return $after_chant;
}

$my_mood = $_GET["my_mood"];
$chant_number = $_GET["my_number"];

$after_chant_mood = Mood_Chant($my_mood, $chant_number);

echo $after_chant_mood;

?>
```

Save this as **mantra.php**, open **moodinput.html**, and Preview.

In this program we let the user select a number. Then inside of the mood_chant function we call Mantra(first parameter, second parameter) where the second parameter is the number the end user chose on the form in the first place. Notice we changed the function Mantra() to accept two parameters.

By adding a **default value** to the parameter **\$the_number**, you made that parameter completely *optional* when you call **Mantra**. To see this in action, try changing the program so that one of the calls to Mantra() has only one parameter being set.

Note

You can have as many parameters and default values as you want in a function. But you have to make sure that the default-valued parameters are at the *end* of the parameter list. Any idea why? Experiment to find out!

Be sure to save your work and hand in your assignments. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Arrays

Have you ever used one of those weekly pill containers? You know, the ones that keep your vitamins or medicine organized for each day? Surely you've at least *seen* one:



This is an excellent representation of this entire lesson—that box is just an **array** of containers with objects in them. Let's get started with a fresh file and take a breather from the monster we've created.

Creating an Array

Open a new PHP file and type this code into CodeRunner:

```
<?php

$names = array("scott", "kendell", "Trish", "Tony", "Mike", "Debra", "Curt");

echo "<pre>";
print_r($names);
echo "</pre>";

?>
```

Save and preview the file:

```
Array
(
    [0] => scott
    [1] => kendell
    [2] => Trish
    [3] => Tony
    [4] => Mike
    [5] => Debra
    [6] => Curt
)
```



You've just *defined* an **array** named **names**, by passing the seven names as parameters to the built-in PHP **array()** construct. If you think in terms of the pill box above—a huge, people-sized pill box—it would look like this:

0	1	2	3	4	5	6
scott	kendell	Trish	Tony	Mike	Debra	Curt

Now, the great thing about arrays is that you can access and mess with any one of the **elements**—names, pills, whatever values are in the boxes—by using the array *keys*. Let's give Mike a call:

Type the following into CodeRunner:

```
<?php

$names = array("scott", "kendell", "Trish", "Tony", "Mike", "Debra", "Curt");

echo "Who is it? ...".$names[4]."<br/>";

echo "<pre>";
print_r($names);
echo "</pre>";

?>
```

Preview this. Did you see Mike's name? All you did here is retrieve the value of the array **element** at the 4th position, or **index**. In this case, you used the index **4** as the **key**. For kicks, let's replace Mike:

Type the following into CodeRunner:

```
<?php

$names = array("scott", "kendell", "Trish", "Tony", "Mike", "Debra", "Curt");

echo $names[4];

$names[4] = "Josh";
echo "Who is it? ...".$names[4]."<br/>";

echo "<pre>";
print_r($names);
echo "</pre>";

?>
```

See, this is why we love arrays. No scope to contend with, just a simple organization of values that we can mess with at will. So now the **\$names** array looks like this:

0	1	2	3	4	5	6
scott	kendell	Trish	Tony	Josh	Debra	Curt

Note

Notice the new, super-handly, built-in function called **print_r**, which prints out an array in a really nice, readable format. With a little experimentation, you can figure out why we used the `<pre>` and `</pre>` tags, too. You can find out more about this function at php.net.

Associative Arrays

If we wanted to represent the pill box in PHP, it would make sense to use the labels that already exist to mark each box for our purposes as well. Here's one way to do it:

Type the following into CodeRunner:

```
<?php

$weekly_pills = array("S" => "vitamin C",
                      "M" => "Echinacea",
                      "T" => "antibiotic",
                      "W" => "calcium",
                      "Th" => "zinc",
                      "F" => "multivitamin",
                      "Sa" => "alka seltzer");

echo "<pre>";
print_r($weekly_pills);
echo "</pre>";

?>
```

Save it as **pills.php** and preview:

```
Array
(
    [S] => vitamin C
    [M] => Echinacea
    [T] => antibiotic
    [W] => calcium
    [Th] => zinc
    [F] => multivitamin
    [Sa] => alka seltzer
)
```



You've just defined an *associative array*. By using the **=>** operator, you've *associated* each array element value to its own index, or *key*, so that you can access it more intuitively. In other words, an associative array is a way of naming each slot of the array. In this case, the slots are named **S**, **M**, **T**, **W**, **Th**, **F**, and **Sa**, respectively. So now we can store and access values in an array based on these names instead of using indices. Experiment with this:

Type the following into CodeRunner:

```
<?php

$weekly_pills = array("S" => "vitamin C",
                      "M" => "Echinacea",
                      "T" => "antibiotic",
                      "W" => "calcium",
                      "Th" => "zinc",
                      "F" => "multivitamin",
                      "Sa" => "alka seltzer");

echo "<pre>";
print_r($weekly_pills);
echo "</pre>";

//assign a new pill to Thursday
$weekly_pills["Th"] = 'aspirin';

//Does Thursday correspond to index 4? Let's see...
$weekly_pills[4] = 'garlic';

//Let's be lazy and see what happens...
$weekly_pills[] = 'glucose';

echo "<pre>";
print_r($weekly_pills);
echo "</pre>";

?>
```

Save and preview this:

```
Array
(
    [S] => vitamin C
    [M] => Echinacea
    [T] => antibiotic
    [W] => calcium
    [Th] => zinc
    [F] => multivitamin
    [Sa] => alka seltzer
)
```

```
Array
(
    [S] => vitamin C
    [M] => Echinacea
    [T] => antibiotic
    [W] => calcium
    [Th] => aspirin
    [F] => multivitamin
    [Sa] => alka seltzer
    [4] => garlic
    [5] => glucose
)
```



By the way, *all* arrays in PHP are associative. Every array value is assigned to a key index, regardless of whether we defined it. When you *don't* define a key index for an element value, PHP automatically assigns a default index to that value for you. Specifically, it assigns the next increment after the highest integer index used. That's why 'glucose' was assigned to the index **5**—we'd already used **4**.

Type the following into CodeRunner:

```
<?php

$months_of_the_year = array(1 => "January", "February", 4 => "April", 3 => "March",
                             "May", "June", "July", "August", "September", 12 => "December",
                             10 => "October", 11 => "November");

echo "<pre>";
print_r($months_of_the_year);
echo "</pre>";

?>
```

Save it as **months.php** and preview it. Play around with it. Become one with array elements and keys. Oh, and don't forget to study your book or php.net for more fun examples.

Creating Multi-Dimensional Arrays

A multi-dimensional array is simply an array of arrays. That is, we can put arrays in for the values of an array which would be a *two-dimensional array*. A three-dimensional array would be an array of arrays of arrays. Ah, nesting. One of PHP's little joys. Let's modify our **pills.php** to see how it works.

Type the following into CodeRunner:

```
<?php

$weekly_pills = array("S" => array("8am" => "vitamin C",
                                   "1pm" => "antibiotic",
                                   "6pm" => "zinc",
                                   "11pm" => "alka seltzer"),
                      "M" => array("8am" => "vitamin C",
                                   "1pm" => "antibiotic",
                                   "6pm" => "zinc",
                                   "11pm" => "alka seltzer"),
                      "T" => array("8am" => "vitamin C",
                                   "1pm" => "antibiotic",
                                   "6pm" => "zinc",
                                   "11pm" => "alka seltzer"),
                      "W" => array("8am" => "vitamin C",
                                   "1pm" => "antibiotic",
                                   "6pm" => "zinc",
                                   "11pm" => "alka seltzer"),
                      "Th" => array("8am" => "vitamin C",
                                   "1pm" => "antibiotic",
                                   "6pm" => "zinc",
                                   "11pm" => "alka seltzer"),
                      "F" => array("8am" => "vitamin C",
                                   "1pm" => "antibiotic",
                                   "6pm" => "zinc",
                                   "11pm" => "alka seltzer"),
                      "Sa" => array("8am" => "vitamin C",
                                   "1pm" => "antibiotic",
                                   "6pm" => "zinc",
                                   "11pm" => "alka seltzer"));

echo "<pre>";
print_r($weekly_pills);
echo "</pre>";

echo "What pill should I pop right now? ...". $weekly_pills["Th"]["6pm"];

?>
```


Save and preview this code. Wow. That's a lot of pills! But it seems that there are enough people taking enough pills that a container indeed exists that represents this *matrix* of dosages:



Creating a *multi-dimensional array* is as simple as nesting the **array()** construct to your heart's content, to create useful representations of just about anything.

Traversing and Manipulating Arrays

Let's have some fun and send a shout-out to everyone in the **\$names** array. Modify **array.php** as shown

We're feeling friendly. Type the following into CodeRunner:

```
<?php

$names = array("scott", "kendell", "Trish", "Tony", "Mike", "Debra", "Curt");

echo "There are ".count($names)." names in the \$names array.<br/>";
for ($i = 0; $i < count($names); $i++) {
    echo "Dialing index ".$i."...";
    echo "Hey there, ".$names[$i]."!<br/>";
}

?>
```

Note

Yet another excellent built-in PHP function is **count()**. We're sure you can guess what it does, but we still encourage you to check it out at php.net.

Preview this code and feel the love:

```
There are 7 names in the $names array.
Dialing index 0...Hey there, scott!!
Dialing index 1...Hey there, kendell!!
Dialing index 2...Hey there, Trish!!
Dialing index 3...Hey there, Tony!!
Dialing index 4...Hey there, Mike!!
Dialing index 5...Hey there, Debra!!
Dialing index 6...Hey there, Curt!!
```



Just by being friendly, you've *traversed* an array. Traversing simply requires that you hopscotch through all the elements of your array and do something with each value. "For" and "while" loops are great for that, especially when you use numerical indices.

Traversing Associative Arrays with **list()** and **each()**

Here's one guarantee: you're going to use arrays a *lot*. You can create, access, traverse, and manipulate arrays fairly easily IF you know exactly *what* is going into them, *how many* elements they have, and *how deep* the nesting goes in every case. But most of the time, you won't know all that. You'll need to work around any gaps with some nifty programming or some great built-in PHP array functions, like **count()**.

For instance, how would you traverse the associative **\$weekly_pills** array? Using numerical counters won't help. But don't worry, you have options. Here's our recommended way to do it:

Type the following into CodeRunner:

```
<?php

$weekly_pills = array("Sunday" => array("8am" => "vitamin C",
    "1pm" => "antibiotic",
    "6pm" => "zinc",
    "11pm" => "alka seltzer"),
    "Monday" => array("8am" => "vitamin C",
    "1pm" => "antibiotic",
    "6pm" => "zinc",
    "11pm" => "alka seltzer"),
    "Tuesday" => array("8am" => "vitamin C",
    "1pm" => "antibiotic",
    "6pm" => "zinc",
    "11pm" => "alka seltzer"),
    "Wednesday" => array("8am" => "vitamin C",
    "1pm" => "antibiotic",
    "6pm" => "zinc",
    "11pm" => "alka seltzer"),
    "Thursday" => array("8am" => "vitamin C",
    "1pm" => "antibiotic",
    "6pm" => "zinc",
    "11pm" => "alka seltzer"),
    "Friday" => array("8am" => "vitamin C",
    "1pm" => "antibiotic",
    "6pm" => "zinc",
    "11pm" => "alka seltzer"),
    "Saturday" => array("8am" => "vitamin C",
    "1pm" => "antibiotic",
    "6pm" => "zinc",
    "11pm" => "alka seltzer"));

while (list($key, $value) = each($weekly_pills)) {
    echo "Here's what you should take on ".$key."<br/>";

    echo "<pre>";
    print_r($value);
    echo "</pre>";
}

?>
```

Save and preview it:

Here's what you should take on Sunday:

```
Array
(
    [8am] => vitamin C
    [1pm] => antibiotic
    [6pm] => zinc
    [11pm] => alka seltzer
)
```

How did you get all that output? Well, there are two built-in functions working together here.

Let's break it down:

```
while (list($key, $value) = each($weekly_pills)) {  
    echo "Here's what you should take on ".$key."<br/>";  
    .  
    .  
    .  
}
```

list() is not really considered a *function*, but a *language construct*, because it doesn't follow the normal "Parameter in/Return value out" function rule. **list()** is simply a shortcut which, when used with the assignment operator (=) and an **array**, assigns each value of that array to the parameter variables within **list()**.

In other words, this:

```
list($parameter1, $parameter2, $parameter3) = array("value1", "value2", "value3")  
;
```

...is the same as this:

```
$parameter1 = "value1";  
$parameter2 = "value2";  
$parameter3 = "value3";
```

Now let's go on to **each()**, which may be even trickier than **list()**. Trickier, because it introduces an aspect of arrays that we haven't discussed until now: the *array cursor*.

Take a look at the graphical representation of **\$names** again:

0	1	2	3	4	5	6
scott	kendell	Trish	Tony	Josh	Debra	Curt

Now, take your mouse cursor and *point* to each box, one by one, starting with the first entry. You've just demonstrated the way an array cursor works: it *points* to array elements. The array cursor always begins by pointing to an array's first element, and stays where it is until moved by a built-in PHP function.

Here's where **each()** comes in:

Type the following into CodeRunner:

```
<?  
$test_array = array("key1" => "value1",  
    "key2" => "value2",  
    "key3" => "value3");  
  
//start with the beginning  
$new_array1 = each($test_array);  
  
echo "<pre>";  
print_r($new_array1);  
echo "</pre>";  
  
?>
```

Save it as **each.php** and preview it:

```

Array
(
    [1] => value1
    [value] => value1
    [0] => key1
    [key] => key1
)

```



As you may have guessed, **each()** takes an array as its parameter. But what you may *not* have guessed is that it also has an array as its return value. Only the array returned is different from the array passed in.

each() uses the array cursor to access the element currently being pointed to by that cursor. This is called the *current element*. In our above example, the current element is the first element of **\$test_array**. After accessing the element, **each()** creates a *new* array with four elements—using the key and value from the current element of the parameter array—and *returns* that array. In our example, we assigned that array to **\$new_array1**. Finally, **each()** *increments* the array cursor so it points to the next element in the array.

Why four elements in the return array? So that the new array can be accessed both numerically AND associatively. The **key** of the parameter array's current element becomes the **value** for two of the new array's elements, accessed by the keys **0** and **"key"**. The **value** of the parameter array's current element becomes the **value** for the other two elements of the new array, accessed by the keys **1** and **"value"**.

Since **list()** can only deal with numerical keys (it ignores associative keys), the four-element return array is especially handy.

Let's put it all together:

```

while (list($key, $value) = each($weekly_pills)) {
    echo "Here's what you should take on ".$key."<br/>";
    .
    .
    .
}

```

In this example, the "while" loop is monitoring the cursor of our **\$weekly_pills** array. We can trust that the loop won't be infinite because of **each()**. The array cursor will eventually reach the end of the array and point to **null**, but each time it loops, the current element's key (in this case, the day of the week) would be assigned to the variable **\$key**. Similarly, the current element's value (in this case, another array containing that day's pills) would be assigned to the variable **\$value**.

Yikes! That's not just tricky, that's downright eye-crossing. When you do get the hang of it though, this little PHP concoction will serve you well, not only with arrays, but with SQL commands and lots of other programming.

Note

As an alternative to using **list()** and **each()** inside the condition of a while loop, check out **foreach()** loops at php.net.

More built-in functions

How do you know if an element exists in an array? What if you need distinct array elements? How about sorting and merging? All these questions can be answered with built-in PHP functions. Like we said earlier, it would take ages to go through them all, but we should definitely go over some of the major ones.

To cap off our intensive array workout, we leave you with a montage of fun PHP functions. Play, experiment, and refer back to your book or to php.net often. Think about how the functions work. Are array cursors used? What are the parameters? What is the function returning?

Finally, think about how you would write your own PHP functions to perform the same tasks. Would you make the same choices as the PHP folks?

Type the following into CodeRunner:

```
<?php

$scotts_phonebook = array("kendell" => "555-1234",
    "Trish" => "555-2345",
    "Tony" => "555-3456",
    "Mike" => "555-4567",
    "Debra" => "555-5678",
    "Curt" => "555-6789");

$kendells_phonebook = array("scott" => "555-7890",
    "Trish" => "555-2345",
    "Tony" => "555-3456",
    "Debra" => "555-5678",
    "Kate" => "555-8901");

//here's a phonebook combining both Scott's and Kendell's contacts, no duplicates

$combined_phonebook = array_unique(array_merge($scotts_phonebook, $kendells_phonebook))
;

echo "<pre> Combined Phonebook:";
print_r($combined_phonebook);
echo "</pre>";

//sort by name - why do you suppose we aren't assigning the return value to anything?

ksort($combined_phonebook);

echo "<pre>Sorted Phonebook:";
print_r($combined_phonebook);
echo "</pre>";

//here's a phonebook containing only mutual friends of Scott and Kendell

$mutual_friends = array_intersect($scotts_phonebook, $kendells_phonebook);

echo "<pre>Mutual Friends:";
print_r($mutual_friends);
echo "</pre>";

//in this custom function called "invite_friend," a phone number is
//called and that friend is invited to a party.

function invite_friend($phone_number, $name) {
    echo "Calling phone number $phone_number...";
    echo "Hello $name! You're invited to a party!<br/>";
}

//Here's a REALLY tricky built-in function we can use to invite ALL friends to the party.
//Careful, this one has lots of rules regarding the second parameter.

array_walk($combined_phonebook, 'invite_friend');

//Finally, generate a random phone number and see if it's in the phonebook.

$random_phonenumber = "555-".strval(rand(1000,9999));

if (in_array($random_phonenumber, $combined_phonebook)) {
    echo "Phone number ".$random_phonenumber." is in the phonebook.";
}
else {
```

```
    echo "Phone number ".$random_phonenumber." is not in the phonebook.";
}

?>
```

Save it as **phonebooks.php** and preview.

Were you able to figure them out? If not, give yourself some time and don't stress—remember, these are functions built by someone else to save time. If any built-in function doesn't suit your purpose, look for another one...or just write one yourself.

Don't forget to Save your work! And be sure to work on the assignments in your syllabus when you're done here. See you at the next lesson...

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Strings

Welcome back. So, you already know that **strings** are one type of PHP **variable**. And you've been using strings throughout the last five lessons with **echo**, the concat operator (**.**), and in all kinds of functions and loops. You've got strings down, baby.

So why spend an entire lesson on the letters, numbers, and symbols that make up strings?

The truth is, we've only explored the tip of the proverbial iceberg when it comes to strings. In fact, they are the cornerstones of many a web-based, database-driven application. Like piranha, you should never underestimate the feisty little critters.

So get your waist-high galoshes on, fire up **PHP** in CodeRunner, and let's get cracking.

What's a String Anyway?

And what is it hiding from us? String *is* its real name, right? Let's see what's going on here. Remember our LAMP acronym?:

Type the following into a new PHP file in CodeRunner:

```
<?php

$lamp_l = "Linux";
$lamp_a = "Apache";
$lamp_m = "MySQL";
$lamp_p = "PHP";

echo "<br/>The stack begins with ".$lamp_l.", and goes on to include "
    ".$lamp_a.", ".$lamp_m.", and ".$lamp_p."!<br/>";

//These supposedly simple strings are hiding something...

echo "Gimme an L!  ".$lamp_l[0]."!<br/>";
echo "Gimme an A!  ".$lamp_a[0]."!<br/>";
echo "Gimme an M!  ".$lamp_m[0]."!<br/>";
echo "Gimme a P!  ".$lamp_p[0]."!<br/>";

?>
```

Save it as **strings.php**, then click Preview. You should see this:

```
The stack begins with Linux, and goes on to include Apache, MySQL, and PHP!
Gimme an L! L!
Gimme an A! A!
Gimme an M! M!
Gimme a P! P!
```



Wait a minute. Why were we just able to use the array operator **[]** to access the first letters of the LAMP acronym? Aha, now the truth comes forth.

That sneaky string doesn't want you to know it has a secret identity. You see, a **string** is a special type of *array*, one where each **character** --letter, number, symbol, newline, whatever takes up one byte of space-- is assigned a numerical key index. Here's what the string "Linux" would look like in our pillbox representation from the arrays lesson:

0	1	2	3	4	5
L	I	N	U	X	Ø

Note The last box you see contains simply the **NULL** character, which has always been used to *terminate* strings in the C language - the language PHP is based upon. ([Check out the history of PHP.](#))

Manipulating Strings

Let's explore strings further. We're going to make a new PHP file called **bologna.php**.

Type the following into CodeRunner:

```
<?php

function spell_me($mystring) {
    $i = 0;
    while ($mystring[$i] != null) {
        if ($i == 0) {
            echo $mystring[$i];
        }
        else {
            echo " - ".$mystring[$i];
        }
        $i++;
    }
}

$string_1 = "bologna";
$string_2 = "oscar";
$string_3 = $string_2;

$string_3[0] = 'm';
$string_3[1] = 'a';
$string_3[2] = 'y';
$string_3[3] = 'e';

//Sing along if you remember the commercial!

echo "My ".$string_1." has a first name, it's ";
spell_me($string_2);
echo "<br/>";

echo "My ".$string_1." has a second name, it's ";
spell_me($string_3);
echo "<br/>";

?>

Oh I love to eat it every day, <br/>
And if you ask me, why I'll say...<br/>

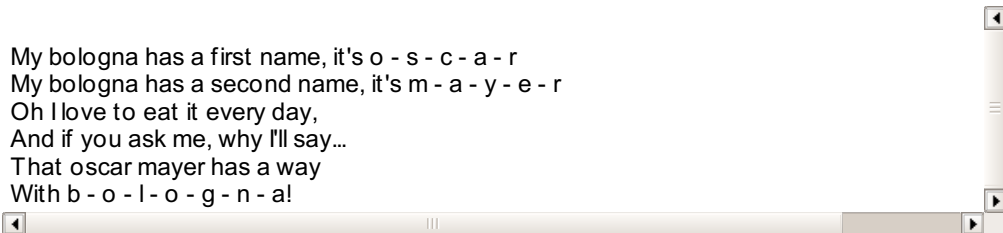
<?

echo "That ".$string_2." ".$string_3." has a way<br/>
    With ";
spell_me($string_1);
echo "!!";

?>
```

Note For reference here's the Oscar Mayer [bologna song](#)

Preview for the lyrics of the song:



My bologna has a first name, it's o - s - c - a - r
My bologna has a second name, it's m - a - y - e - r
Oh I love to eat it every day,
And if you ask me, why I'll say...
That oscar mayer has a way
With b - o - l - o - g - n - a!

Through the power of commercial jingles, we're able to uncover two more truths about strings: not only can we *access* the characters within a string using the `[]` operator, but we can use the same operator to *traverse* and *manipulate* strings, just like arrays.

Take another look:

```
function spell_me($mystring) {
    $i = 0;
    while ($mystring[$i] != null) {
        if ($i == 0) {
            echo $mystring[$i];
        }
        else {
            echo " - ".$mystring[$i];
        }
        $i++;
    }
}

$string_1 = "bologna";
$string_2 = "oscar";
$string_3 = $string_2;

$string_3[0] = 'm';
$string_3[1] = 'a';
$string_3[2] = 'y';
$string_3[3] = 'e';

.
.
.
```

In our function `spell_me()`, we used a **while loop** to *traverse* the string parameter, stopping when we reached the null character. Then we *manipulated* `$string_3` by *assigning* new characters to the indices we wanted to change. In no time, "oscar" turned to "mayer," and all were spelled correctly.

Go ahead, keep humming the tune - we don't mind.

Other nifty string shortcuts

Type the BLUE stuff into your document in CodeRunner:

```
<?php

function spell_me($mystring) {
    $i = 0;
    while ($mystring[$i] != null) {
        if ($i == 0) {
            echo $mystring[$i];
        }
        else {
            echo " - ".$mystring[$i];
        }
        $i++;
    }
}

$string_1 = "bologna";
$string_2 = "oscar";
$string_3 = $string_2;

$string_3[0] = 'm';
$string_3[1] = 'a';
$string_3[2] = 'y';
$string_3[3] = 'e';

//Sing along if you remember the commercial!

echo "My $string_1 has a first name, it's "; //we took out the concat operators
spell_me($string_2);
echo "<br/>";

echo "My $string_1 has a second name, it's ";
spell_me($string_3);
echo "<br/>";

?>

Oh I love to eat it every day, <br/>
And if you ask me, why I'll say...<br/>

<?

echo "That $string_2 $string_3 has a way<br/>
    With ";
spell_me($string_1);
echo "!";

?>
```

Preview this. Nothing changed, right? This is a cool shortcut created especially for strings in PHP, called *embedding variables*. Since you'll most likely use PHP in web pages, you can thank us later for showing you this shortcut. It provides a more intuitive method of creating and outputting dynamic strings without the need for all those annoying concat operators and quotation marks.

There's only one small complication with this shortcut. What happens if you want to display an actual dollar sign (\$) along with all the embedded variables?

Type the following into CodeRunner:

```
<?php

function spell_me($mystring) {
    $i = 0;
    while ($mystring[$i] != null) {
        if ($i == 0) {
            echo $mystring[$i];
        }
        else {
            echo " - ".$mystring[$i];
        }
        $i++;
    }
}

$string_1 = "bologna";
$string_2 = "oscar";
$string_3 = $string_2;

$string_3[0] = 'm';
$string_3[1] = 'a';
$string_3[2] = 'y';
$string_3[3] = 'e';

//Sing along if you remember the commercial!

echo "My $string_1 has a first name, it's "; //we took out the concat operators
spell_me($string_2);
echo "<br/>";

echo "My $string_1 has a second name, it's ";
spell_me($string_3);
echo "<br/>";

?>

Oh I love to eat it every day, <br/>
And if you ask me, why I'll say...<br/>

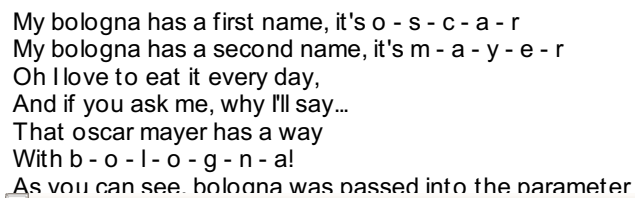
<?

echo "That $string_2 $string_3 has a way<br/>
    With ";
spell_me($string_1);
echo "!";

echo "<br/>As you can see, $string_1 was passed into the parameter $mystring.";

?>
```

You should get something like this:



My bologna has a first name, it's o - s - c - a - r
My bologna has a second name, it's m - a - y - e - r
Oh I love to eat it every day,
And if you ask me, why I'll say...
That oscar mayer has a way
With b - o - l - o - g - n - a!
As you can see, bologna was passed into the parameter

Well, that's a bunch of bologna. While we wanted to output the actual variable names, the **echo** command tried to replace them with their values instead. This happens anytime echo sees a dollar sign (\$) followed by something that could pass as a variable name.

How can we stop it? *Escape* it.

Type the following into CodeRunner:

```
<?php

function spell_me($mystring) {
    $i = 0;
    while ($mystring[$i] != null) {
        if ($i == 0) {
            echo $mystring[$i];
        }
        else {
            echo " - ".$mystring[$i];
        }
        $i++;
    }
}

$string_1 = "bologna";
$string_2 = "oscar";
$string_3 = $string_2;

$string_3[0] = 'm';
$string_3[1] = 'a';
$string_3[2] = 'y';
$string_3[3] = 'e';

//Sing along if you remember the commercial!

echo "My $string_1 has a first name, it's "; //we took out the concat operators
spell_me($string_2);
echo "<br/>";

echo "My $string_1 has a second name, it's ";
spell_me($string_3);
echo "<br/>";

?>

Oh I love to eat it every day, <br/>
And if you ask me, why I'll say...<br/>

<?

echo "That $string_2 $string_3 has a way<br/>
    With ";
spell_me($string_1);
echo "!";

echo "<br/>As you can see, \$string_1 was passed into the parameter \$mystring."
;

?>
```

You should get this:

```
My bologna has a first name, it's o - s - c - a - r
My bologna has a second name, it's m - a - y - e - r
Oh I love to eat it every day,
And if you ask me, why I'll say...
That oscar mayer has a way
With b - o - l - o - g - n - a!
As you can see, $string_1 was passed into the parameter $mystring.
```

Ah, much better. Just by adding a little backslash (\) before each dollar sign (\$), we're able to tell PHP that we really do want the name itself displayed, not the value.

That backslash is handy for *escaping* several other characters too. Refer to your book or to php.net to embed them all into your subconscious.

Built-in String Functions

"String functions?" you say, "I don't need no stinking string functions. I could use all the built-in array functions on strings too!"

While that may be true in C, PHP treats strings as a different **type** with its own set of built-in functions, generally keeping their secret identity under wraps. Try using an array function to traverse a string:

Type the following into CodeRunner:

```
<?php

function spell_me($mystring) {
    for ($i = 0; $i < count($mystring); $i++) {
        if ($i == 0) {
            echo $mystring[$i];
        }
        else {
            echo " - ".$mystring[$i];
        }
    }
}

$string_1 = "bologna";
$string_2 = "oscar";
$string_3 = $string_2;

$string_3[0] = 'm';
$string_3[1] = 'a';
$string_3[2] = 'y';
$string_3[3] = 'e';

//Sing along if you remember the commercial!

echo "My $string_1 has a first name, it's "; //we took out the concat operators
spell_me($string_2);
echo "<br/>";

echo "My $string_1 has a second name, it's ";
spell_me($string_3);
echo "<br/>";

?>

Oh I love to eat it every day, <br/>
And if you ask me, why I'll say...<br/>

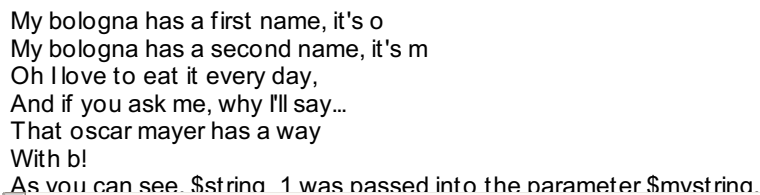
<?

echo "That $string_2 $string_3 has a way<br/>
    With ";
spell_me($string_1);
echo "!";

echo "<br/>As you can see, \"$string_1\" was passed into the parameter \"$mystring.\"";

?>
```

Preview it and you should get this:



```
My bologna has a first name, it's o
My bologna has a second name, it's m
Oh I love to eat it every day,
And if you ask me, why I'll say...
That oscar mayer has a way
With b!
As you can see, \"$string_1\" was passed into the parameter \"$mystring.\"
```

Not exactly the catchiest lyrics anymore. Now try it with a built-in string function.

Type the following into CodeRunner:

```
<?php

function spell_me($mystring) {
    for ($i = 0; $i < strlen($mystring); $i++) {
        if ($i == 0) {
            echo $mystring[$i];
        }
        else {
            echo " - ".$mystring[$i];
        }
    }
}

$string_1 = "bologna";
$string_2 = "oscar";
$string_3 = $string_2;

$string_3[0] = 'm';
$string_3[1] = 'a';
$string_3[2] = 'y';
$string_3[3] = 'e';

//Sing along if you remember the commercial!

echo "My $string_1 has a first name, it's "; //we took out the concat operators
spell_me($string_2);
echo "<br/>";

echo "My $string_1 has a second name, it's ";
spell_me($string_3);
echo "<br/>";

?>

Oh I love to eat it every day, <br/>
And if you ask me, why I'll say...<br/>

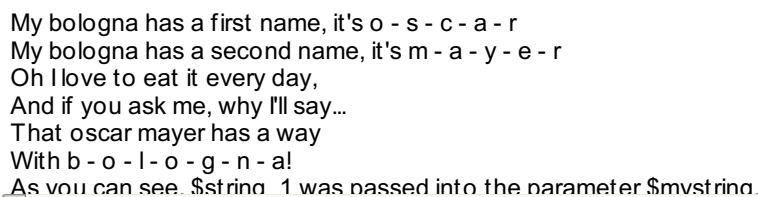
<?

echo "That $string_2 $string_3 has a way<br/>
    With ";
spell_me($string_1);
echo "!";

echo "<br/>As you can see, \"$string_1\" was passed into the parameter \"$mystring.\"";

?>
```

You should get this:



```
1 My bologna has a first name, it's o - s - c - a - r
2 My bologna has a second name, it's m - a - y - e - r
3 Oh I love to eat it every day,
4 And if you ask me, why I'll say...
5 That oscar mayer has a way
6 With b - o - l - o - g - n - a!
7 As you can see, $string_1 was passed into the parameter $mystring.
8
9
10
```

See, it's not such a bad thing. Having specialized string functions means they'll be faster, easier, and more intuitive to use.

Soon we'll be working with HTML forms and dynamic inputs, which really flex the muscles of built-in PHP string functions. However, even without forms, string functions have thousands of uses. Here we have peppered our oscar mayer song with a plethora of useful string functions. Play, experiment, and refer back to your book or to php.net as much as you need. Try out the code below. Can you figure out how they all work?

Type the following into CodeRunner:

```
<?php

function spell_me($mystring) {
    for ($i = 0; $i < strlen($mystring); $i++) {
        if ($i == 0) {
            echo strtoupper($mystring[$i]);
        }
        else {
            echo " - ".strtoupper($mystring[$i]);
        }
    }
}

$string_1 = "bologna";
$string_2 = "oscar mayer";
$space_index = strpos($string_2, " ");

//let's spell boloney how we really say it...
echo "My ".str_replace('gna','ney',$string_1)." has a first name, it's ";
spell_me(substr($string_2,0,$space_index));
echo "<br/>";

echo "My $string_1 has a second name, it's ";
spell_me(substr($string_2,$space_index+1)); //notice this has only two parameters
echo "<br/>";

?>

Oh I love to eat it every day, <br/>
And if you ask me, why I'll say...<br/>

<?

//we're tired of putting in the HTML <br/> tags...
echo "That $string_2 ".nl2br("has a way
    With ");
spell_me($string_1);
echo "!";

$sale_price = 1; //a dollar
echo "<br/>On sale for ".number_format($sale_price, 2)."!";

?>
```

Before we move on, experiment with these questions in CodeRunner:

- Does it matter whether you use single quotation marks (') or double quotation marks (") with strings?
- Can you mix the two types of quotation marks? Do you have to escape them if you do?
- Are there any built-in array functions that *do* work with strings?
- Would you have built the `substr()` function differently?

Regular Expressions

Not many subjects can make a programmer groan like that of **regular expressions**. They are immensely useful, yes - they are used to create "wildcard" strings so that you can, say, verify that someone has entered a valid email address or a correct phone number format. However, learning "Regex" patterns can sometimes feel as though you're deciphering the Rosetta Stone. Even the ever-helpful php.net pawns you off to a cryptic "man page" when dealing with Regex rules. Aargh.

But hey! We've got "learning by doing" on our side. And when we learn by doing, we can accomplish anything.

Type the following into a new PHP file in CodeRunner:

```
<?php

//here's a simple function to check Regex patterns against string parameters

function check_regex($myregex, $mystring) {
    if (preg_match("/$myregex/", $mystring)) {
        echo "The pattern '$myregex' is found in $mystring.<br/>";
    }
    else {
        echo "The pattern '$myregex' is NOT found in $mystring.<br/>";
    }
}

$regex_1 = "log";

$string_1 = "bologna";
check_regex($regex_1, $string_1);

?>
```

You should get:

The pattern 'log' is found in bologna.



Here we have an example of a **regular expression** -- a simple string: "log." And by using the built-in PHP function `preg_match()`, we are simply checking to see if a "log" is found in "bologna." Of course it is. Notice the quotes and forward slashes needed around the `$myregex` variable. These are needed because `preg_match` is a PERL style regex matching function and regex's in PERL must have forward slashes. See what happens if you remove the slashes.

So, you may wonder why we didn't just use the built-in string function `strpos()`. We could have. But here's where it gets interesting...

The plot thickens. Type the following into CodeRunner:

```
<?php

//here's a simple function to check Regex patterns against string parameters

function check_regex($myregex, $mystring) {
    if (preg_match("/$myregex/", $mystring)) {
        echo "The pattern '$myregex' is found in $mystring.<br/>";
    }
    else {
        echo "The pattern '$myregex' is NOT found in $mystring.<br/>";
    }
}

$regex_1 = "log$";

$string_1 = "bologna";
check_regex($regex_1, $string_1);

$string_2 = "catalog";
check_regex($regex_1, $string_2);

?>
```

Preview this:

The pattern 'log\$' is NOT found in bologna.
The pattern 'log\$' is found in catalog.



Now THIS result we could not get from `strpos`. Since when is a dollar sign (\$) found in the word "catalog"?

As it turns out, the dollar sign has a special meaning in regular expressions, and it's different from the meaning it usually has for PHP variables. In regular expressions, placing a dollar sign (\$) after a string means *"at the end of the string"*.

Take another look:

```
<?php
.
.
.

$regex_1 = "log$";

$string_1 = "bologna";
check_regex($regex_1, $string_1);

$string_2 = "catalog";
check_regex($regex_1, $string_2);

?>
```

Because we specified **\$regex_1** to be **"log\$"** and not just **"log"**, our function **check_regex()** now checks to see if **"log"** is found at the **end** of each of our strings. This is why it returned **true** for "catalog," but **false** for "bologna."

This is the key to regular expressions. More than just a random string tool, regular expressions are an entirely different language for creating and comparing strings with very specific patterns in mind: the presence of specific characters, the number of occurrences of each character, and their location in the string. In this case, we were concerned with the location of the string "log." Let's try another one...

Type the following into CodeRunner:

```
<?php

//here's a simple function to check Regex patterns against string parameters

function check_regex($myregex, $mystring) {
    if (preg_match("/$myregex/", $mystring)) {
        echo "The pattern '$myregex' is found in $mystring.<br/>";
    }
    else {
        echo "The pattern '$myregex' is NOT found in $mystring.<br/>";
    }
}

$regex_1 = "^cat";

$string_1 = "concatenate";
check_regex($regex_1, $string_1);

$string_2 = "catalog";
check_regex($regex_1, $string_2);

?>
```

The pattern '^cat' is NOT found in concatenate.

The pattern '^cat' is found in catalog.



You guessed it. Placing a carat (^) in front of your Regex string means *"at the beginning of the string."*

Character Ranges and Number of Occurrences

Type the following into CodeRunner:

```
<?php

//here's a simple function to check Regex patterns against string parameters

function check_regex($myregex, $mystring) {
    if (preg_match("/$myregex/", $mystring)) {
        echo "The pattern '$myregex' is found in $mystring.<br/>";
    }
    else {
        echo "The pattern '$myregex' is NOT found in $mystring.<br/>";
    }
}

$regex_1 = "cat.*a";

$string_1 = "concatenate";
check_regex($regex_1, $string_1);

$string_2 = "catalog";
check_regex($regex_1, $string_2);

?>
```

Preview this:

The pattern 'cat.*a' is found in concatenate.
The pattern 'cat.*a' is found in catalog.



Whoa. That is a crazy Regex pattern. Yet it was found in the strings "concatenate" AND "catalog." What gives?

And speaking of concatenate, isn't that period (.) the concatenate operator in PHP? Not this time. Just like the dollar sign (\$), the period (.) has a much different meaning when it comes to regular expressions. In this case, it represents *any* character, like a wildcard.

As for the asterisk (*), that means "zero or more". Put it all together, and the regular expression "**cat.*a**" means *"the string 'cat,' followed by zero or more characters, followed by an 'a'"*.

Is that found in "**concatenate**"? Yes: the string '**cat**' is found, followed by two characters '**e**' and '**n**', followed by an '**a**'. How about "**catalog**"? Yep: '**cat**' is followed by zero characters, followed by an '**a**'. Let's try a REALLY crazy Regex:

Type the following into CodeRunner:

```
<?php

//here's a simple function to check Regex patterns against string parameters

function check_regex($myregex, $mystring) {
    if (preg_match("/$myregex/", $mystring)) {
        echo "The pattern '$myregex' is found in $mystring.<br/>";
    }
    else {
        echo "The pattern '$myregex' is NOT found in $mystring.<br/>";
    }
}

$regex_1 = "cat(a|e)+[a-z]{2,5}";

$string_1 = "concatenate";
check_regex($regex_1, $string_1);

$string_2 = "catalog";
check_regex($regex_1, $string_2);

$string_3 = "catamaran";
check_regex($regex_1, $string_3);

$string_4 = "scathing";
check_regex($regex_1, $string_4);

$string_5 = "pontificates";
check_regex($regex_1, $string_5);

?>
```

Preview this:

The pattern 'cat(a|e)+[a-z]{2,5}' is found in concatenate.
The pattern 'cat(a|e)+[a-z]{2,5}' is found in catalog.
The pattern 'cat(a|e)+[a-z]{2,5}' is found in catamaran.
The pattern 'cat(a|e)+[a-z]{2,5}' is NOT found in scathing.
The pattern 'cat(a|e)+[a-z]{2,5}' is NOT found in pontificates.



Now, this may seem overwhelming, but it's actually just a series of simple Regex patterns. Let's break them down:

cat(a|e)+[a-z]{2,5}

- **(a|e)**: The pipe character (|) in regular expressions means **OR**, so in this case we're looking for "either an 'a' or an 'e'". Parentheses (|) are used to separate out expressions when we are *nesting* them, just like always.
- **+**: The plus sign (+) is just like the asterisk (*), except it's looking for **one or more** of the characters it follows. Since we preceded it with the expression (a|e), in this case it means "one or more of either 'a' or 'e'".
- **[a-z]**: To allow entire ranges of characters as a shortcut, we use square brackets ([]) and the dash (-). So in this case, we're looking for "any lowercase letter from 'a' to 'z'".
- **{2,5}**: Curly braces ({}) are used like the plus sign and asterisk, indicating a range of occurrences of the preceding expression. In this case, because {2,5} follows [a-z], we're looking for "2 to 5 occurrences of any lowercase letter from 'a' to 'z'".

Put it all together, and we find that the pattern **cat(a|e)+[a-z]{2,5}** in Regex-speak means "The string 'cat', followed by **one or more** 'a's or 'e's, followed by **at least 2, but not more than 5** lowercase letters, from 'a' to 'z'."

Can you figure out why it's not found in "scathing" or "pontificates"?

Excluding Characters

Now, if you thought THAT was confusing, consider this: What if you had the all-important task of, say, removing funky characters from a file name and replacing them with something benign? Here's where things REALLY get screwy.

Type the following into CodeRunner:

```
<?php

//here's a simple function to check Regex patterns against string parameters

function check_regex($myregex, $mystring) {
    if (preg_match("/$myregex/", $mystring)) {
        echo "The pattern '$myregex' is found in $mystring.<br/>";
    }
    else {
        echo "The pattern '$myregex' is NOT found in $mystring.<br/>";
    }
}

//here's a function that takes in a file name, and replaces all funky characters
with an underscore "_"

function clean_filename($file_name) {
    $bad_characters = "[^a-zA-Z0-9.]";
    $new_filename = preg_replace("/$bad_characters/", "_", $file_name);
    return $new_filename;
}

$bad_filename = "file[3*1 name.doc";

$good_filename = clean_filename($bad_filename);

echo "'$bad_filename' has been changed to '$good_filename'.";

?>
```

Preview this:

'file[3*1 name.doc' has been changed to 'file_3_1_name.doc'.



We know, we know, this makes no sense at all. First of all, the carat (^) is supposed to mean "at the beginning of the string." The period (.) is supposed to represent a wildcard character. And what's with the ranges of characters -- a-z, A-Z, and 0-9 -- stuck together like that? Groan.

Well, as it turns out, when it comes to whatever's in the square brackets ([]), the rules change. Let's take a closer look at brackets in regular expressions.

[^a-zA-Z0-9.]

- **^**: When used within square brackets, the caret (^) *negates* everything after it - just like the exclamation point(!) in PHP. So in this case it's looking for characters that DON'T match what's inside the brackets.
- **a-zA-Z0-9**: Everything within square brackets comes together to represent only one character. Therefore, characters placed within the brackets are treated as if a pipe character (|) was inserted in between each one. For instance, [abcd] is the same as (a|b|c|d), and in this case, a-zA-Z0-9 is the same as ([a-z]|[A-Z]|[0-9]), or more simply, "any alphanumeric character".
- **.**: Within square brackets, every character except for the caret(^), the dash(-), and the right bracket itself(]) is taken as a literal character - including the period(.) that would normally be considered a wildcard character.

Put it all together, and we find that the pattern **[^a-zA-Z0-9.]** actually means "any character which is NOT an alphanumeric character or a period."

Escaping Characters

Regular Expressions are extremely useful in PHP - especially since you'll be doing a lot of HTML form processing. For instance, how can you ensure that someone's entered their phone number properly?

Type the following into CodeRunner:

```
<?php

//here's a simple function to check Regex patterns against string parameters

function check_regex($myregex, $mystring) {
    if (preg_match("/$myregex/", $mystring)) {
        echo "The pattern '$myregex' is found in $mystring.<br/>";
    }
    else {
        echo "The pattern '$myregex' is NOT found in $mystring.<br/>";
    }
}

//here's a function that takes in a file name, and replaces all funky characters
with an underscore "_"

function clean_filename($file_name) {
    $bad_characters = "[^a-zA-Z0-9.]";
    $new_filename = preg_replace("/$bad_characters/", "_", $file_name);
    return $new_filename;
}

//here's a function which validates an American phone number

function validate_phone($phone_number) {
    $good_phone = "^(?:[0-9]{3})?(?:[0-9]{3})(?:[0-9]{4})$";

    if (preg_match("/$good_phone/", $phone_number)) {
        echo "The phone number is valid.<br/>";
    }
    else echo "The phone number is NOT valid.<br/>";
}

$phone_number1 = "34x.d98.1123";
validate_phone($phone_number1);

$phone_number2 = "(217) 555-1212";
validate_phone($phone_number2);

?>
```

Preview this:

34x.d98.1123 is NOT valid.
(217) 555-1212 is valid.



Let's break this down: **^(?:[0-9]{3})?(?:[0-9]{3})(?:[0-9]{4})\$**

- **^**: Since we're outside any square brackets, the caret (^) takes on its original meaning—"at the beginning of the string." By the same token, we use the dollar sign (\$) to mean "at the end of the string," so that we have an exact match.
- **(?:[0-9]{3})?**: Some Americans use parentheses (()) to enclose the 3-digit area code of their phone numbers. To allow this possibility, we use the question mark (?) much like the asterisk or plus sign, only this time to denote "zero or one" of the leading parenthesis (()). However, because parentheses usually mean *nesting*, we use a backslash (\) to escape the character. This must always be done when not within the square brackets. The same is true with **(?:[0-9]{3})**.
- **[0-9]{3}**: Since the area code of the American phone number system uses exactly 3 digits, we use **{3}** to require exactly 3 of any digit, denoted by **[0-9]**. We use the same logic with the 3-digit prefix, as well as the ending 4 digits of the phone number.

- **([-\])** Usually between the area code and the prefix of a phone number, folks use either a space (" "), a dash (-), or a period(.). Therefore, we use the parentheses(**()**) along with the pipe character (**|**) to say "*either a space or a dash or a period.*" We put a backslash before the period because we must escape it. An unescaped period matches a single character without caring what that character is. So since we want a literal period, we add the backslash to "escape" the character having that special meaning.

In case your eyes are crossing right now, remember that **regular expressions** take a lot of patience, practice, and trial-and-error to get right. Refer back to this lesson, to books you may have, or to the web, often. [Here's a great article on regular expressions in PHP.](#)

Whew! We've covered a lot of ground. Don't forget to **Save** your work and hand in your **assignments** from your syllabus. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Fixing Broken PHP



This lesson is all about frustration. Frustration that comes in the form of **parse errors**, **infinite loops**, **unmatched brackets**, and **logical mistakes**. Frustration that makes your face grimace and your fingers type furiously, pounding as if sheer force could will the keys into assembling proper PHP code from the mangled mess that is your own program. Ah yes, we know this feeling well.

In previous lessons we focused on the basics of PHP, keeping examples and projects to finite blocks of code. We're sure you've handled the frustrating errors like a trooper so far. In the upcoming lessons, we'll begin constructing more complex programs to solve real-world problems, which means the threat of frustration looms larger than ever. You're going to need some serious ammo for creating and debugging scalable programs. Your sanity just may

depend on it.

So let's take a break from new PHP concepts and focus on technique for a while. Got CodeRunner in **PHP** syntax? Good - let's get going.

Things Professors Don't Talk About Enough

We're guilty of it too. We introduce you to concepts that theoretically work just fine, assuming that everything typed in just so, and that we've explained the concept perfectly. So of *course* these concepts will work perfectly for you, every time you apply them, right?

Let's find out using the following silly program we assembled from concepts covered in previous lessons.

It's okay to copy and paste, JUST THIS ONCE! Paste this into CodeRunner:

```
<?
function mantra($the_sound,$the_number = 10) {
    $chant = 1;
    while ($chant <= $the_number;) {
        echo $the_sound..." ";
    }
}

function Mood_Chant($my_mood){
    if ($my_mood == "happy") {
        mantra("OM");
        $after_chant = "<br/>I feel serene now.";
    }
    else if ($my_mood == "sad") {
        mantra("okay");
        $after_chant = "<br/>I feel better now.";
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
        $after_chant = "<br/>Ahhh, much better.  I've calmed down now.";
    }
    else if ($my_outlook == "indifferent") {
        mantra("Wake up");
        $after_chant = "<br/>I'm awake now.";
    }
    else {
        mantra("Try harder");
        $after_chant = "<br/>I'll try harder now.";
    }
    return $after_chant;
}

function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $my_cash) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $my_cash ) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $my_cash && $cereal_prices['Fruit_Loops']
] < $my_cash ){
        $mood = "angry";
        echo "Fine!  I'll get some Fruit Loops.";
    }
    else {
        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}

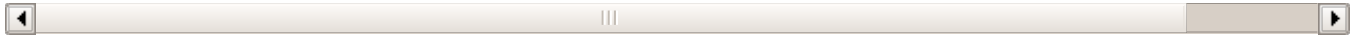
<h3>The emotional roller-coaster of buying breakfast cereal</h3>
<?
$cereal_prices = array('Captain_Crunch' => 5, 'Fruit_Loops' => 3);
$my_cash = 4;
?>
I have $<? echo $my_cash; ?> in my pocket, and I want to buy some Captain Crunch!<br/>
<ul>
<li>Captain Crunch costs $<? echo $cereal_prices['Captain_Crunch']; ?></li>
<li>Fruit Loops costs $<? echo $cereal_prices['Fruit_Loops']; ?></li>
</ul>

$my_mood = whats_my_emotion($cereal_prices, $my_cash);
if (!($my_mood == "sad")) {
```

```
$after_chant_mood = Mood_Chant($my_mood, $chant_number);  
}  
echo "<br/>".$after_chant_mood;  
  
?>
```

Preview this:

Parse error: syntax error, unexpected ';' in /users/certjosh/useractivepreviewtmp123.php line 4



Okay, WRONG. Even a benign-looking program like the above works beautifully in theory, but *never* in practice -- at least for the first hundred times you try it. Trust us. Do you think our examples worked perfectly the first time we wrote them? Hardly.

But even with this relatively small amount of code, where do you begin to debug it? Here is where you're usually on your own...

...but hey, not in this lesson! We're not too far removed from our humble beginnings to know how hard it is to master debugging. Consider this a support group for frustrated coders, and you're invited.

Debugging Tips

Utilizing Error Messages

Let's Preview again:

Parse error: syntax error, unexpected ';' in /users/certjosh/useractivepreviewtmp line 4



If you're lucky, you'll get an easy to see error message right away, like you're getting now. In other situations you may have to ask your system administrator where she keeps the PHP error logs. In any case, the First Rule of Debugging is: *check the error messages first*. They may seem cryptic at times, but they almost always give you the information you need. In particular, the **line number** where the problem is located.

Since our error message indicated **line 4**, go to that line. What do you see?

Suddenly, our **parse error** is as loud as a mariachi band. There shouldn't be a semicolon (;) inside the parentheses (()) of a **while loop**! This is easy enough to fix: just remove the semicolon (;).


Good for you! You fixed the error, and now everything should work perfectly, right?

Riddle-Me-This Error Messages

Cross your fingers and Preview again:

Parse error: syntax error, unexpected '<' in /users/certjosh/useractivepreviewtmp line 53



Yikes, another error message! Mild frustration ensues. Well, no problem, we'll just do the same thing we did before. But this time try the Go To Line  button. Type in line **53**:

Hmm, that's strange. This isn't even PHP code, it's HTML code -- and *perfect* HTML code, at that. Why would PHP single out a line of good HTML code in its error message?

We're going to have to look around for more clues, which brings us to the Second Rule of Debugging: *Check lines closest to the error message second*. Let's give that a shot, by looking at line **52**:

And there you have it - a tiny curly bracket (**}**), indicative of PHP code. Were you able to solve the riddle of the error message? We forgot to *delimit* the PHP with a **?>** between the PHP code and the HTML code, so the PHP parser was attempting to read the HTML code as PHP! Hence the message: "unexpected '<' on line **53**". It didn't know any better.

Go ahead and add the delimiter (**?>**), and you have squashed another bug in our program. Let's hope that's the last one.

Errors without Error Messages

Chant 'no error messages' three times, then Preview again:

The emotional roller-coaster of buying breakfast cereal

I have \$4 in my pocket, and I want to buy some Captain Crunch!

- Captain Crunch costs \$5
- Fruit Loops costs \$3

```
$my_mood = whats_my_emotion($cereal_prices, $my_money); if (!$my_mood == "
$after_chant_mood = Mood_Chant($my_mood, $chant_number); } echo "
".$after_chant_mood; ?>
```



Hey, the chant worked - no error messages! But wait - there's still an error. Looks like "no error messages" is not the same as "no errors". Which brings us to the Third Rule of Debugging: *When there are no error messages, check your output*. Or, just work on your chant.

Look at your Preview again. It seems that the trouble starts right after the statement: *"Fruit Loops costs \$3"*. After that, chaos erupts. So let's take a look at our code now, and try to pinpoint the problem.

Pay attention to the errors we already fixed, and where the new error seems to be happening:

```
<?
function mantra($the_sound,$the_number = 10) {
    $chant = 1;
    while ($chant <= $the_number) { //first we took out the rogue semicolon...
        echo $the_sound..." ";
    }
}

function Mood_Chant($my_mood){
    if ($my_mood == "happy") {
        mantra("OM");
        $after_chant = "<br/>I feel serene now.";
    }
    else if ($my_mood == "sad") {
        mantra("okay");
        $after_chant = "<br/>I feel better now.";
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
        $after_chant = "<br/>Ahhh, much better.  I've calmed down now.";
    }
    else if ($my_outlook == "indifferent") {
        mantra("Wake up");
        $after_chant = "<br/>I'm awake now.";
    }
    else {
        mantra("Try harder");
        $after_chant = "<br/>I'll try harder now.";
    }
    return $after_chant;
}

function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $my_cash) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $my_cash ) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $my_cash && $cereal_prices['Fruit
_Loops'] < $my_cash ){
        $mood = "angry";
        echo "Fine!  I'll get some Fruit Loops.";
    }
    else {
        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
//then we added the delimiter here...
?>
<h3>The emotional roller-coaster of buying breakfast cereal</h3>
<?
$cereal_prices = array('Captain_Crunch' => 5, 'Fruit_Loops' => 3);
$my_cash = 4;
?>
I have $<? echo $my_cash; ?> in my pocket, and I want to buy some Captain Crunch
!<br/>
<ul>
<li>Captain Crunch costs $<? echo $cereal_prices['Captain_Crunch']; ?></li>
<li>Fruit Loops costs $<? echo $cereal_prices['Fruit_Loops']; ?></li>
</ul>
```

```
//BUT NOW THE PROBLEM APPEARS TO BE HERE

$my_mood = whats_my_emotion($cereal_prices, $my_cash);
if (!($my_mood == "sad")) {
    $after_chant_mood = Mood_Chant($my_mood, $chant_number);
}
echo "<br/>".$after_chant_mood;

?>
```

Looking at the code, we see now that we've done it again - we've forgotten a **delimiter**, this time an opening delimiter (<?). Why didn't we get an error message like before? Because this time the code went from HTML to PHP - so it was HTML attempting to render the PHP code, not the other way around. HTML is more forgiving in this sense, and simply prints out the code.

Be sure to add the delimiter <?. Are we done now?

Logical Errors

Signs point to no:

The emotional roller-coaster of buying breakfast cereal

I have \$4 in my pocket, and I want to buy some Captain Crunch!

- Captain Crunch costs \$5
- Fruit Loops costs \$3

Oh well. I'm going home.

At first glance, everything appears to be correct. No error messages, no garbled output. But before you breathe that sigh of relief, remember that this silly program is supposed to determine our mood and purchasing behavior, based on cereal prices and how much money we have.

Take another look at the code:

```
function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $my_cash) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $my_cash ) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $my_cash && $cereal_prices['Fruit
_Loops'] < $my_cash){
        $mood = "angry";
        echo "Fine! I'll get some Fruit Loops.";
    }
    else {
        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
```

From our output, we see that we have \$4 -- not enough to buy Captain Crunch for \$5, but enough to buy Fruit Loops for \$3. In our program, that's supposed to make us **angry**, but we'd rather invoke a calming mantra chant and buy Fruit Loops anyway. So why are we dejected and going home?

This is called a **logical error**, and unfortunately it seems that the output isn't helping us much in the way of

clues to fix it. Which brings us to the Fourth Rule of Debugging: *When the output doesn't show the error, create strategic output that does.*

Type the following green code into CodeRunner:

```
<?
function mantra($the_sound,$the_number = 10) {
    $chant = 1;
    while ($chant <= $the_number) { //first we took out the rogue semicolon...
        echo $the_sound."... ";
    }
}

function Mood_Chant($my_mood){
    if ($my_mood == "happy") {
        mantra("OM");
        $after_chant = "<br/>I feel serene now.";
    }
    else if ($my_mood == "sad") {
        mantra("okay");
        $after_chant = "<br/>I feel better now.";
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
        $after_chant = "<br/>Ahhh, much better.  I've calmed down now.";
    }
    else if ($my_outlook == "indifferent") {
        mantra("Wake up");
        $after_chant = "<br/>I'm awake now.";
    }
    else {
        mantra("Try harder");
        $after_chant = "<br/>I'll try harder now.";
    }
    return $after_chant;
}

function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $my_cash) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $my_cash ) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $my_cash && $cereal_prices['Fruit
_Loops'] < $my_cash ){
        $mood = "angry";
        echo "Fine!  I'll get some Fruit Loops.";
    }
    else {
        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
//then we added the delimiter here...
?>
<h3>The emotional roller-coaster of buying breakfast cereal</h3>
<?
$cereal_prices = array('Captain_Crunch' => 5, 'Fruit_Loops' => 3);
$my_cash = 4;
?>
I have $<?
//We know it's not here, because the output has been correct
echo $my_cash; ?> in my pocket, and I want to buy some Captain Crunch!<br/>
<ul>
<li>Captain Crunch costs $<? echo $cereal_prices['Captain_Crunch']; ?></li>
<li>Fruit Loops costs $<? echo $cereal_prices['Fruit_Loops']; ?></li>
```

```

</ul>

<?
//then we added the delimiter here...

$my_mood = whats_my_emotion($cereal_prices, $my_cash);
//Let's add some echo statements to figure out our logic.
echo "\$my_mood is $my_mood";

if (!($my_mood == "sad")) {
    $after_chant_mood = Mood_Chant($my_mood, $chant_number);
}
echo "<br/>".$after_chant_mood;

?>

```

Preview this:

The emotional roller-coaster of buying breakfast cereal

I have \$4 in my pocket, and I want to buy some Captain Crunch!

- Captain Crunch costs \$5
- Fruit Loops costs \$3

Oh well, I'm going home.\$my_mood is sad

So we find that when the function **whats_my_emotion()** returns, its value is **sad**, not angry. Since we know the values of **\$my_cash** and **\$cereal_prices** are correct, it looks like we've narrowed the problem down to **whats_my_emotion()**. Now what do we do?

Let's add some more echo statements - but this time use them *within* **whats_my_emotion()**, just to see what happens.

Add the following green code into CodeRunner:

```
<?
function mantra($the_sound,$the_number = 10) {
    $chant = 1;
    while ($chant <= $the_number) { //first we took out the rogue semicolon...
        echo $the_sound."... ";
    }
}

function Mood_Chant($my_mood){
    if ($my_mood == "happy") {
        mantra("OM");
        $after_chant = "<br/>I feel serene now.";
    }
    else if ($my_mood == "sad") {
        mantra("okay");
        $after_chant = "<br/>I feel better now.";
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
        $after_chant = "<br/>Ahhh, much better.  I've calmed down now.";
    }
    else if ($my_outlook == "indifferent") {
        mantra("Wake up");
        $after_chant = "<br/>I'm awake now.";
    }
    else {
        mantra("Try harder");
        $after_chant = "<br/>I'll try harder now.";
    }
    return $after_chant;
}

function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $my_cash) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $my_cash ) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $my_cash && $cereal_prices['Fruit
_Loops'] < $my_cash ){
        $mood = "angry";
        echo "Fine!  I'll get some Fruit Loops.";
    }
    else {
        //We know the output is coming from here, so let's add echo statements:
        echo "Within whats_my_emotion, \$cereal_prices:<pre>";
        print_r($cereal_prices);
        echo "\$my_cash = ".$my_cash."</pre>";

        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
//then we added the delimiter here...
?>
<h3>The emotional roller-coaster of buying breakfast cereal</h3>
<?
$cereal_prices = array('Captain_Crunch' => 5, 'Fruit_Loops' => 3);
$my_cash = 4;
?>
```

```

I have $<?
//We know it's not here, because the output has been correct
echo $my_cash; ?> in my pocket, and I want to buy some Captain Crunch!<br/>
<ul>
<li>Captain Crunch costs $<? echo $cereal_prices['Captain_Crunch']; ?></li>
<li>Fruit Loops costs $<? echo $cereal_prices['Fruit_Loops']; ?></li>
</ul>

<?
//then we added the delimiter here...

$my_mood = whats_my_emotion($cereal_prices, $my_cash);
//Let's add some echo statements to figure out our logic.
echo "\$my_mood is $my_mood";

if (!($my_mood == "sad")) {
    $after_chant_mood = Mood_Chant($my_mood, $chant_number);
}
echo "<br/>".$after_chant_mood;

?>

```

Preview this:

The emotional roller-coaster of buying breakfast cereal

I have \$4 in my pocket, and I want to buy some Captain Crunch!

- Captain Crunch costs \$5
- Fruit Loops costs \$3

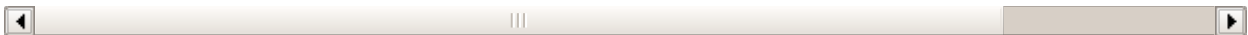
Within `whats_my_emotion`, `$cereal_prices`:

```

Array
(
    [Captain_Crunch] => 5
    [Fruit_Loops] => 3
)
$my_cash =

```

Oh well, I'm going home. `$my_mood` is sad



This is starting to look pretty messy, but it does tell us everything we need to know. Through our `echo` and `print_r` statements, we can see that the **logical error** is definitely *within* **`whats_my_emotion()`**. Why? Because the parameter **`$my_cash`** was never properly passed in -- causing the resulting mood to be **sad** instead of **angry**.

Take a closer look at `whats_my_emotion()`:

```
function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $my_cash) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $my_cash ) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $my_cash && $cereal_prices['Fruit
_Loops'] < $my_cash ){
        $mood = "angry";
        echo "Fine! I'll get some Fruit Loops.";
    }
    else {
        //We know the output is coming from here, so let's add echo statements:
        echo "Within whats_my_emotion, \$cereal_prices:<pre>";
        print_r($cereal_prices);
        echo "\$my_cash = ".$my_cash."</pre>";

        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
```

Looking at our **if/else statements**, we can see that we make all kinds of comparisons between `$my_cash` and the various cereal prices, yet no matter what we set `$my_cash` to before we pass it to `whats_my_emotion()`, it comes up blank *within* `whats_my_emotion()`. And then the error becomes clear: *within* `whats_my_emotion()`, the parameter should be called `$cash_money`, NOT `$my_cash`!

And so, Sherlock, it looks like we have solved the mystery of the **logical error**. We can now *remove* the extraneous echo and print_r statements and fix the problem, once and for all.

Type the following green code into CodeRunner:

```
<?
function mantra($the_sound,$the_number = 10) {
    $chant = 1;
    while ($chant <= $the_number) { //first we took out the rogue semicolon...
        echo $the_sound..." ";
    }
}

function Mood_Chant($my_mood){
    if ($my_mood == "happy") {
        mantra("OM");
        $after_chant = "<br/>I feel serene now.";
    }
    else if ($my_mood == "sad") {
        mantra("okay");
        $after_chant = "<br/>I feel better now.";
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
        $after_chant = "<br/>Ahhh, much better.  I've calmed down now.";
    }
    else if ($my_outlook == "indifferent") {
        mantra("Wake up");
        $after_chant = "<br/>I'm awake now.";
    }
    else {
        mantra("Try harder");
        $after_chant = "<br/>I'll try harder now.";
    }
    return $after_chant;
}

function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $cash_money) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $cash_money ) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $cash_money && $cereal_prices['Fruit_Loops'] < $cash_money ){
        $mood = "angry";
        echo "Fine!  I'll get some Fruit Loops.";
    }
    else {
        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
//then we added the delimiter here...
?>
<h3>The emotional roller-coaster of buying breakfast cereal</h3>
<?
$cereal_prices = array('Captain_Crunch' => 5, 'Fruit_Loops' => 3);
$my_cash = 4;
?>
I have $<? echo $my_cash; ?> in my pocket, and I want to buy some Captain Crunch
!<br/>
<ul>
<li>Captain Crunch costs $<? echo $cereal_prices['Captain_Crunch']; ?></li>
<li>Fruit Loops costs $<? echo $cereal_prices['Fruit_Loops']; ?></li>
</ul>
```

```
<?
//then we added the delimiter here...

$my_mood = whats_my_emotion($cereal_prices, $my_cash);
if (!( $my_mood == "sad" )) {
    $after_chant_mood = Mood_Chant($my_mood, $chant_number);
}
echo "<br/>".$after_chant_mood;

?>
```

Infinite Loops, Infinite Headaches

Preview this:

The emotional roller-coaster of buying breakfast cereal

I have \$4 in my pocket, and I want to buy some Captain Crunch!

- Captain Crunch costs \$5
- Fruit Loops costs \$3

Fine! I'll get some Fruit Loops.mississippi... mississippi... mississippi... mississippi... mis
mississippi... mississippi... mississippi... mississippi... mississippi... mississippi... mississip
mississippi... mississippi... mississippi... mississippi... mississippi... mississippi... mississip
mississippi... mississippi... mississippi... mississippi... mississippi... mississippi... mississip
mississippi... mississippi... mississippi... mississippi... mississippi... mississippi... mississip
mississippi... mississippi... mississippi... mississippi... mississippi... mississippi... mississip

YIKES, MAKE IT STOP! This is one of the worst errors of all: **infinite loops**. It causes memory leaks in your computer, aching in your head, and it may very well have crashed your entire browser...we hope that wasn't the case.

This brings us to the Fifth Rule of Debugging: *Always end your loops!*

Take a look at our while loop:

```
function mantra($the_sound,$the_number = 10) {
    $chant = 1;
    while ($chant <= $the_number) {
        echo $the_sound."... ";
    }
}
```

The good news is, in our case it's easy to see what went wrong. The while loop is set to end when `$chant` is **greater than \$the_number**, which defaults to 10. But we never increased `$chant`. Let's fix it!

Type the following into CodeRunner:

```
<?
function mantra($the_sound,$the_number = 10) {
    $chant = 1;
    while ($chant <= $the_number) { //first we took out the rogue semicolon...
        echo $the_sound."... ";
        $chant++;
    }
}

function Mood_Chant($my_mood){
    if ($my_mood == "happy") {
        mantra("OM");
        $after_chant = "<br/>I feel serene now.";
    }
    else if ($my_mood == "sad") {
        mantra("okay");
        $after_chant = "<br/>I feel better now.";
    }
    else if ($my_mood == "angry") {
        mantra("mississippi");
        $after_chant = "<br/>Ahhh, much better.  I've calmed down now.";
    }
    else if ($my_outlook == "indifferent") {
        mantra("Wake up");
        $after_chant = "<br/>I'm awake now.";
    }
    else {
        mantra("Try harder");
        $after_chant = "<br/>I'll try harder now.";
    }
    return $after_chant;
}

function whats_my_emotion($cereal_prices, $cash_money) {
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $cash_money) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $cash_money) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $cash_money && $cereal_prices['Fruit_Loops'] < $cash_money){
        $mood = "angry";
        echo "Fine!  I'll get some Fruit Loops.";
    }
    else {
        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
//then we added the delimiter here...
?>
<h3>The emotional roller-coaster of buying breakfast cereal</h3>
<?
$cereal_prices = array('Captain_Crunch' => 5, 'Fruit_Loops' => 3);
$my_cash = 4;
?>
I have $<? echo $my_cash; ?> in my pocket, and I want to buy some Captain Crunch
!<br/>
<ul>
<li>Captain Crunch costs $<? echo $cereal_prices['Captain_Crunch']; ?></li>
<li>Fruit Loops costs $<? echo $cereal_prices['Fruit_Loops']; ?></li>
```

```

</ul>

<?
//then we added the delimiter here...

$my_mood = whats_my_emotion($cereal_prices, $my_cash);
if (!($my_mood == "sad")) {
    $after_chant_mood = Mood_Chant($my_mood, $chant_number);
}
echo "<br/>".$after_chant_mood;

?>

```

Preview this:

The emotional roller-coaster of buying breakfast cereal

I have \$4 in my pocket, and I want to buy some Captain Crunch!

- Captain Crunch costs \$5
- Fruit Loops costs \$3

Fine! I'll get some Fruit Loops.mississippi... mississippi... mississippi... mississippi... mississippi
mississippi... mississippi... mississippi... mississippi... mississippi...

Ahhh, much better. I've calmed down now.

Ahh, much better. And although it was quite the ordeal, we're all the better for it. Pat yourself on the back and raise your glass to stress relief through Debugging!

Notes on Scalable Programming

Now that you know the Rules of Debugging, you're almost ready to put them to the test by building some large-scale projects. However, before you go on, it's important to stress some very important points to help reduce *your* stress.

Before you Code, Pseudocode

What's **pseudocode**? Just a little jot-down of your program logic in English (or whatever your native language may be). Like this:

Here's how we might pseudocode `whats_my_emotion()`:

```

If I have enough money to buy both cereals,
    My mood is happy.
Otherwise, if I have enough money to buy Captain Crunch,
    My mood is indifferent.
Otherwise, if I can't buy Captain Crunch, but can buy Fruit Loops,
    My mood is angry.
Otherwise, I'm sad no matter what.

```

Pseudocode is a way for you to organize your thoughts and design your logic before you start coding your program. Think of it as a blueprint for your software development. Using pseudocode, you can take a look at the big picture and catch any flaws in your design--*before* they cause you a week's worth of debugging. Plus, you can refer back to it as you go to ensure that you're sticking to your original design.

Make your Program Readable

What if we had coded `whats_my_emotion()` like this?

```
function what_is_it($a, $b = false) {
    if ($a >= $b) {
        $c = "happy";
    }
    else if ($d[0] < $b) {
        $c = "indifferent";
    }
    else if ($d[0] > $b && $d[1] < $b) {
        $c = "angry";
    }
    else {
        $c = "sad";
    }
    return $c;
}
```

Sure, we know exactly what it means at the time we write it, but when we go back later, we might not have a clue what any of it means, rendering it essentially worthless. And by the way, if you write code like this, forget ever getting promoted - you won't find anyone who can decipher your code enough to take over your lower position. You'd be stuck with it, buddy.

So just be sure to use readable, intuitive variable, and function names all the time, every time. If you find yourself slipping into using vague names, just remember what we told you about promotion. That should snap you back into shape.

Comment Until You're Blue in the Face

By the same token, you can kick your program's readability up a notch by using comments whenever you can. Use them to help recall what you've done, or to indicate to other programmers what your functions do. That's why they're there after all.

In particular, it's imperative that you start off each program, and every function within it, with a synopsis of the functions it performs, parameters it takes, and what it returns.

Like this:

```
function whats_my_emotion($cereal_prices, $cash_money) {
    #whats_my_emotion returns an emotion of happy, indifferent, angry or sad base
    d upon
    #two parameters, $cereal_prices -- an array of floats -- and float $cash_mone
    y.
    $total = array_sum($cereal_prices); //array_sum is a built-in PHP function
    if ($total < $cash_money) {
        $mood = "happy";
        echo "I'll buy both Captain Crunch and Fruit Loops!";
    }
    else if ($cereal_prices['Captain_Crunch'] < $cash_money) {
        $mood = "indifferent";
        echo "I'll buy Captain Crunch.";
    }
    else if ($cereal_prices['Captain_Crunch'] > $cash_money && $cereal_prices['Fr
    uit_Loops'] < $cash_money) {
        $mood = "angry";
        echo "Fine! I'll get some Fruit Loops.";
    }
    else {
        $mood = "sad";
        echo "Oh well, I'm going home.";
    }
    return $mood;
}
```

This will become more and more important as you build more reusable code, and even libraries which can be

shared by others--either within your company, or within the **open-source community**.

Code in Bite-Size Chunks

You'll notice throughout these lessons that our chosen process of learning to build programs is extremely repetitious - we typed a bit of code, *Previewed* it, added a little more code, *Previewed* it, and so on.

If you program one small part of your code at a time, you'll be much less likely to be overwhelmed with bugs and logical errors when it comes time to test. This is where your pseudocode can help as well, by showing you where you can divide your large program into smaller, "bite-size" chunks to make it more manageable.

Debug as You Work

There's nothing worse than writing a HUGE amount of code, only to find it's a complete mess. As long as you *Preview* often, you'll catch bugs as you go along, which will make your life much easier in the long run.

Reuse Functions as Much as Possible

What if we had written several different functions instead of one **whats_my_emotion()**, or simply copied and pasted the same code throughout our program? Instead of fixing the code once, we would have had to deal with it over and over again.

The biggest argument for creating **functions** for anything and everything: if something goes wrong with the code, you only have to debug it **once**, then every time it's called, it works.

Always create a function for *any finite task*, even if you're not sure you'll use it more than once. You'll be surprised at how useful it will become as you continue programming.

Utilize Available Resources

As if we haven't been preaching it enough: we live in an age where information is *always* available. There are reference books, Safari accounts, and web sites like PHP.net. Use them. And if you can't find your answer, there are communities of millions of PHP programmers just like yourself you can consult. Don't be afraid to ask questions!

Can you believe how far you've come? So far you've learned all the basics of PHP you need to get going with some meaty web projects. And from now on, that's exactly what you're going to do.

Don't forget to **Save** your work and hand in your **assignments** from your syllabus. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Forms in PHP

As promised, you're about to put the material you've learned into robust, real-world applications. Until now, there has been only one thing missing from your skillset preventing this: **user input**.

PHP was created specifically to work with the internet - to make the web surfer's life easier by customizing his experience, and to make the programmer's life easier by making that customization convenient for her. But without a way to gather information from the web surfer, all the convenience and power of PHP is worthless. What good is customization if the user's needs aren't met?

We are able to gather user input through a little HTML tag called `<form>`. Since we'll be using HTML and PHP in tandem, be prepared to use **both HTML and PHP syntax**. Let's go!

Forms Review

Start with an HTML form. Make sure you're using **HTML** syntax, and TYPE the following:

CODE TO TYPE:

```

<body>
<h3>Contact ACME Corporation</h3>
<form method="POST" action="contact.php">
<table>
<tr>
<td align="right">
Name:
</td>
<td align="left">
<input type="text" size="25" name="name" value="">
</td>
</tr>
<tr>
<td align="right">
Email:
</td><td align="left">
<input type="text" size="25" name="email" value="">
</td>
</tr>
<tr>
<td align="right">
Type of Request:
</td>
<td align="left">
<select name="whoami">
<option value="" />Please choose...
<option value="newcustomer" />I am interested in becoming a customer.
<option value="customer" />I am a customer with a general question.
<option value="support" />I need technical help using the website.
<option value="billing" />I have a billing question.
</select>
</td>
</tr>
<tr>
<td align="right">
Subject:
</td>
<td align="left">
<input type="text" size="50" max="50" name="subject" value="">
</td>
</tr>
<tr>
<td align="right" valign="top">
Message:
</td>
<td align="left">
<textarea name="message" cols="50" rows="8">
</textarea>
</td>
</tr>
<tr>
<td colspan="2" align="left">
How did you hear about us?
<ul>
<input type="radio" name="found" value="wordofmouth" />Word of Mouth<br/>
<input type="radio" name="found" value="search" />Online Search<br/>
<input type="radio" name="found" value="article" />Printed publication/article<br/>
<input type="radio" name="found" value="website" />Online link/article<br/>
<input type="radio" name="found" value="other" />Other
</ul>
</td>
</tr>
<tr>
<td colspan="2">
<input type="checkbox" name="update1" checked="checked">Please email me updates about y

```

```

our products.<br/>
<input type="checkbox" name="update2">Please email me updates about products from third
-party partners.
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="SUBMIT" />
</td></tr>
</table>
</form>
</body>

```

You'll see this:

Contact ACME Corporation

Name:

Email:

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third-party partners.

Look familiar? This is a simple contact form, where a user can inquire about the services on a website and give a little information about him/herself. Also noteworthy is that it contains all the major form types: **text**, **text area**, **select**, **radio**, **checkbox**, and **submit**.

Each form element has a **name** attribute and a **value** attribute, except for **text area**, which has an ending tag instead of a **value** attribute. Furthermore, **radio** buttons all have the same name to ensure only one is checked, while **checkboxes** have different names so that any of them can be checked. **select** tags contain their names and values within separate **option** tags, for that nice drop-down-menu effect.

Note

You mean it *doesn't* look familiar? We're assuming this is review for you - if you're completely lost, you may want to take a look at our [HTML and CSS](#) course.

It's a nice-looking form, but if you want something done with that information, you're going to have to create a PHP script to process the input. Go ahead and **Save your form** -- you can name it **contact.html**.

Using Superglobals to Read Form Inputs

Now, let's switch CodeRunner to **PHP** and start a new file.

In PHP, type the following:

```
<?php

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: " . $_POST['name'] . "<br/>";
echo "Email: " . $_POST['email'] . "<br/>";
echo "Type of Request: " . $_POST['whoami'] . "<br/>";
echo "Subject: " . $_POST['subject'] . "<br/>";
echo "Message: " . $_POST['message'] . "<br/>";
echo "How you heard about us: " . $_POST['found'] . "<br/>";
echo "Update you about our products: " . $_POST['update1'] . "<br/>";
echo "Update you about partners' products: " . $_POST['update2'] . "<br/>";

?>
```

Preview this:

Thank You!

Here is a copy of your request:

Name:

Email:

Type of Request:

Subject:

Message:

How you heard about us:

Update you about our products:

Update you about partners' products:



Well, that didn't do much good. And what's this `$_POST[]` array anyway??

But wait, there's more. **Save** this PHP file and call it **contact.php**. Now, switch back to **HTML** in CodeRunner, where you should still have your **contact.html** file ready.

Preview this, and fill in the form:

Contact ACME Corporation

Name:

Email:

Type of Request: I need technical help using the website.

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☒ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third-party partners.

SUBMIT

Now, within your Preview window, click SUBMIT. What did you get?

Hopefully you got something like this:

Thank You!

Here is a copy of your request:

Name: Trish

Email: trish@myemail.com

Type of Request: support

Subject: Please help!

Message: I can't get the darn thing to work!

How you heard about us: website

Update you about our products: on

Update you about partners' products:

So why did it work this time? Here's where the magic of that `$_POST[]` array is revealed.

Take another look at the form tag in contact.html:

```
<form method="POST" action="contact.php">
```

If you remember, forms themselves can be submitted using several different methods - two of the most important methods are **GET** and **POST**. If you've ever programmed a web application in a different language - say Perl or C - you might also remember using complicated CGI libraries to extract form data from either the **query string** in the case of the GET method, or from the **environment variables** in the case of the POST method.

However, because PHP was created with the web in mind, this process has been greatly simplified, using special variables called **superglobals**.

Let's look at contact.php again:

```
<?php

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$_POST['name']."<br/>";
echo "Email: ".$_POST['email']."<br/>";
echo "Type of Request: ".$_POST['whoami']."<br/>";
echo "Subject: ".$_POST['subject']."<br/>";
echo "Message: ".$_POST['message']."<br/>";
echo "How you heard about us: ".$_POST['found']."<br/>";
echo "Update you about our products: ".$_POST['update1']."<br/>";
echo "Update you about partners' products: ".$_POST['update2']."<br/>";

?>
```

Did you notice something familiar about the key indices of `$_POST[]` -- **name**, **email**, **whoami**, etc.? You see, PHP does all the work for you here - it processes the form input and places all the values into the **superglobal array** `$_POST[]`, an associative array with the key indices corresponding to the form element names. This is done automatically, whenever a form is submitted using the **POST method**, and the array works in any scope - that's why it's called a **superglobal** variable.

Note `$_POST`). By convention, we don't normally use the underscore at the beginning of variable names (as in `$_POST`). However, they are used in **superglobals** to prevent any clashing with your own variable names.

What do you do if you use the **GET method** in your form? Experiment with this and find out. If you need help, check out php.net.

Extracting Superglobals into Variables

As if the **superglobal** variables weren't easy enough, PHP goes even further to make reading form inputs easy for you.

In PHP, change contact.php with the following blue code:

```
<?php

extract($_POST, EXTR_PREFIX_SAME, "post");

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";
echo "Update you about our products: ".$update1."<br/>";
echo "Update you about partners' products: ".$update2."<br/>";

?>
```

Save **contact.php** again, then go back to **contact.html** and Preview. What did you get?

Preview contact.html and submit the form like before:

Thank You!

Here is a copy of your request:

Name: Trish
Email: trish@myemail.com
Type of Request: support
Subject: Please help!
Message: I can't get the darn thing to work!
How you heard about us: website
Update you about our products: on
Update you about partners' products:



Wow - it worked even without the `$_POST[]` array! This is yet another simplification in the process that can be done through the built-in function `extract()`. In this case, the form elements passed through the `$_POST[]` array have been extracted into PHP variables, accessible by anything within the program. We indicated to `extract()` that we wanted the PHP variable names to correspond to the form element names by passing in the flag `EXTR_PREFIX_SAME` as a parameter. You can read more about `extract()` here: <http://www.php.net/manual/en/function.extract.php>.

Note

In previous versions of PHP, a php config directive called **register globals** automatically created global variables from GET and POST form elements. However, many dangers arose in using register globals, and as a result, PHP has removed them from PHP 5 and newer versions.

Superglobals are brilliant innovations in web programming - all built into PHP to make your world an easier place to live. Not to mention *our* world - did you notice just how *short* this lesson is? Exactly.

Nesting Variable Names

Just one more cool feature before we move on...

In PHP, change contact.php with the following, in blue:

```
<?php

extract($_POST, EXTR_PREFIX_SAME, "post");

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

?>
```

Preview contact.html and submit the form like before:

Thank You!

Here is a copy of your request:

Name: Trish
Email: trish@myemail.com
Type of Request: support
Subject: Please help!
Message: I can't get the darn thing to work!
How you heard about us: website
update1: on
update2:



Did you see what happened there? We were able to dynamically construct the name of our "update#" form elements through a **for loop**, and then *access the value* of that element through the variables we created with **extract()**. This was done by **nesting variable names**.

Take another look:

```
for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": "; << evaluates to "update1" or "update2"
    echo $$element_name; << evaluates to $update1 or $update2, whose values are "on" or
    "off"
    echo "<br/>";
}
```

Nesting variable names is just like all the nesting we did in previous lessons - first **\$element_name** is evaluated, and then that value is used to evaluate the nested **\$(element_name)**. Name nesting can be done with ALL variables, however, it's especially useful when you create dynamic form names and then need to read them with the variables passed in through **superglobals** and **extract()**. It's definitely worth learning this handy trick.

We're just getting warmed up with forms. Don't forget to **Save** your work and hand in your **assignments** from your syllabus. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Utilizing Internet Tools

In the last lesson, we created a contact form for customers to communicate with the customer support department of a corporation. But it's not quite ready for prime time yet. So far, we have no way of knowing what kind of computer or browser the customer is using, no way to catch incomplete form entries, and no way to, well, send the message out.

It's time to fix this! Fire up CodeRunner and open the two files we were working on before: **contact.html** and **contact.php**.

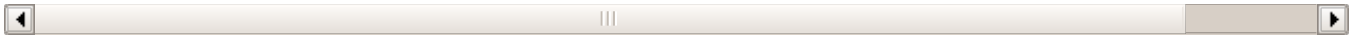
Environment and Server Variables

Preview contact.html and submit the form like before:

Thank You!

Here is a copy of your request:

Name: Trish
Email: trish@myemail.com
Type of Request: support
Subject: Please help!
Message: I can't get the darn thing to work!
How you heard about us: website
update1: on
update2:



Have you ever received a customer support request like this? We have. It's more common than you may think, and it can leave you scratching your head—this customer can't get the *darn* website to work, yet leaves the details of the problem to your mind-reading skills.

Let's look into our crystal ball...

In PHP, change contact.php as shown in blue:

```
<?php

extract($_POST, EXTR_PREFIX_SAME, "post");

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWARDED_FOR'];

?>
```

Switch to contact.html, Preview, and submit the form like before:

Thank You!

Here is a copy of your request:

Name: Trish
Email: trish@myemail.com
Type of Request: support
Subject: Please help!
Message: I can't get the darn thing to work!
How you heard about us: website
update1: on
update2:
You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/4.1 (KHTML, like Gecko) Safari/4.17.9.2
The IP address of the computer you're working on is 63.171.219.74



When it comes to customer support, just as important as the customer's request is knowing where the customer is coming from—perhaps geographically, but more importantly in the sense of what operating system (Windows, Mac) and browser (Safari, Internet Explorer, Firefox) they're using.

Luckily, the folks who worked on our very first web browsers way back in the day, already thought of this. They created something called **CGI (Common Gateway Interface) Environment Variables**, which tell us a lot about both the **client**—that's the customer's computer—as well as the **server** -- that's the computer where your PHP script resides (in our case, it's sitting in Champaign, Illinois).

Take another look at this code:

```
echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];  
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWARDED_FOR'];
```

You guessed it—`$_SERVER[]` is another **superglobal array** in PHP. That underscore(_) at the beginning tends to give it away. The information that `$_SERVER[]` holds? **Environment variables** like `HTTP_USER_AGENT` and `HTTP_X_FORWARDED_FOR`. But what do they mean?

Now take another look at the output:

```
You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/418 (KHTML, like Gecko) Safari/417.9.2  
The IP address of the computer you're working on is 63.171.219.74
```

Not so luckily, the folks who created the environment variables didn't make them easy to decipher. Here's a little translation for the two we're using:

- **HTTP_USER_AGENT** gives you information about the computer and web software your customer is using. Since you're testing your own form, it should be telling you what computer and web software *you're* using. The format is something like this: **operating system/version (more info) web library/version (more info) web browser/version (more info)**.

In our case, we got the information **Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/418 (KHTML, like Gecko) Safari/417.9.2**. Very cryptically, this tells us that we are on a Macintosh computer with a Mac OS X operating system, using the Safari web browser. Obviously, *your* result will most likely be different from this—you might be on a Windows XP computer, for instance, using Internet Explorer (MSIE). [Here](#) is a list of more browsers than you'll ever care to know, and their HTTP_USER_AGENT translations. Can you find yours?

- **HTTP_X_FORWARDED_FOR** gives you the **IP address** of either your customer's computer, or if your customer is using an Internet Service Provider like AOL, the IP address of one of its servers. What's an **IP (Internet Protocol) address**? Every computer on the internet has one -- a unique identifier, chosen within the Internet Protocol Standard. It's useful to know, because it can indicate the customer's country of origin, through any [Whois tool](#). More importantly, if it has been determined that a particular customer is a fraud, there are ways to block the IP address from ever getting to your site—something you will learn in a later course.

There are lots of useful **environment variables**. [Here](#) is a very useful list to reference.

Using HTTP Headers

Another important issue in customer support—and really any interface that requires form input—is ensuring that all fields are properly filled in. How can you help a customer if he doesn't include his email address or contact info? But of course he'll include it, right? You'd be surprised.

In PHP, change contact.php with the following blue code:

```
<?php

#We used the superglobal $_POST here
if (!($_POST['name'] && $_POST['email'] && $_POST['whoami']
    && $_POST['subject'] && $_POST['message'])) {
    echo "Please make sure you've filled in all required information.";
    exit();
}

extract($_POST, EXTR_PREFIX_SAME, "post");

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWARDED_FOR'];

?>
```

Note

Notice the new built-in PHP function we used here: **exit()**. This essentially stops the program in its tracks. How's that for lack of commitment?

Switch to contact.html, Preview, and submit the form like before—only this time, try leaving something blank:

Please make sure you've filled in all required information.



So essentially when someone leaves something blank, we're letting them know about it. But now the customer has to go back to the form and find out what's wrong. What if we could take them back to the form automatically? Let's give it a shot:

In PHP, change contact.php with the following, in blue:

```
<?php

#We used the superglobal $_POST here
if (!($_POST['name'] && $_POST['email'] && $_POST['whoami']
    && $_POST['subject'] && $_POST['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $url = "http://".$_SERVER['HTTP_HOST']."/contact.html";
    header("Location: ".$url);
    exit();
}

extract($_POST, EXTR_PREFIX_SAME, "post");

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name." : ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWARDED_FOR'];

?>
```

Switch to contact.html, Preview your form, and submit it, leaving something blank. You should get this:

Contact ACME Corporation

Name:

Email:

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third-party partners.

SUBMIT

Whoa! What just happened? If you left something blank on your form and submitted it, you just got the same form back, blank again.

Let's take another look at this code:

```
$url = "http://".$_SERVER['HTTP_HOST']."/contact.html";  
header("Location: ".$url);  
exit();
```

You may already know that all HTML-based web pages use the **HyperText Transfer Protocol (HTTP)** to render properly in your web browser—that's why you always see <http://> at the beginning of every web address. But what you may NOT know is that before any HTML is rendered on your web browser, a series of invisible **headers** are passed so that your browser knows exactly what to do with the code. Most of these headers are pretty obscure, but a few are extremely useful. [Click here](#) for a reference.

Of course, since PHP *embeds* HTML within its code, it can also manipulate **HTTP headers** through the built-in function **header()**. In this case, we were able to set the header **"Location: "** with the URL of the contact form [contact.html](#) we created. As a result of sending that header, the browser *redirected* the user back to the form.

Note Any headers that are sent using **header()** must come **BEFORE** any PHP or HTML output. Otherwise, the browser will get confused, and next thing you know, you're debugging.

You'll also notice we used another **environment variable**, called **HTTP_HOST**. This variable returns the **domain name** of the web address where your [contact.php](#) script resides.

In our case, our domain name, or HTTP_HOST, is josh.onza.net. It is a live web site, on the internet for everyone to see: <http://josh.onza.net/>. Pretty lame web site, huh? Keep this in mind when you create your website using your own domain name—you could have a lame web site like us, or you could have a professional online portfolio to show to all your friends, colleagues, and potential employers when you apply for your first LAMP-based programming job.

Manipulating Query Strings

But we digress. And in the meantime, simply redirecting our poor customer to a blank form is a horrible way to treat someone who's already frustrated with the website. There has to be a more user-friendly way to ask the customer to fix a form field before we submit it.

The problem is, since **contact.html** is a static HTML page, we can't dynamically add anything to it—that's why it's blank. And simply giving the error message "Please fix this" to the customer, like we did before, isn't user-friendly either. What we need is a way to show the customer, nicely, exactly what he needs to fix on the form, without losing any of the answers he's already filled in.

We can do this by *converting* the HTML form into a PHP script of its own. What you need to do is **Save contact.html in PHP syntax, but sure to call it "contact_form.php"**. Or if you'd rather, just copy and paste the HTML code into a new PHP file.

Be sure you're in PHP, and add the following blue code to contact_form.php:

```
<body>
<h3>Contact ACME Corporation</h3>
<form method="GET" action="contact.php">
<table>
<tr>
<td align="right">
Name:
</td>
<td align="left">
<input type="text" size="25" name="name" value="<? echo $_GET['name']; ?>" />
</td>
</tr>
<tr>
<td align="right">
Email:
</td><td align="left">
<input type="text" size="25" name="email" value="<? echo $_GET['email']; ?>" />
</td>
</tr>
<tr>
<td align="right">
Type of Request:
</td>
<td align="left">
<select name="whoami">
<option value="" />Please choose...
<option value="newcustomer"<?
if ($_GET['whoami'] == "newcustomer") {
    echo " selected";
}
?> />I am interested in becoming a customer.
<option value="customer"<?
if ($_GET['whoami'] == "customer") {
    echo " selected";
}
?> />I am a customer with a general question.
<option value="support"<?
if ($_GET['whoami'] == "support") {
    echo " selected";
}
?> />I need technical help using the website.
<option value="billing"<?
if ($_GET['whoami'] == "billing") {
    echo " selected";
}
?> />I have a billing question.
</select>
</td>
</tr>
<tr>
<td align="right">
Subject:
</td>
<td align="left">
<input type="text" size="50" max="50" name="subject" value="<? echo $_GET['subject']; ?>" />
</td>
</tr>
<tr>
<td align="right" valign="top">
Message:
</td>
<td align="left">
<textarea name="message" cols="50" rows="8">
<? echo $_GET['message']; ?>
</td>
</tr>
</table>
</form>
</body>
```



```

</textarea>
</td>
</tr>
<tr>
<td colspan="2" align="left">
How did you hear about us?
<ul>
<input type="radio" name="found" value="wordofmouth" />Word of Mouth<br/>
<input type="radio" name="found" value="search" />Online Search<br/>
<input type="radio" name="found" value="article" />Printed publication/article<br/>
<input type="radio" name="found" value="website" />Online link/article<br/>
<input type="radio" name="found" value="other" />Other
</ul>
</td>
</tr>
<tr>
<td colspan="2">
<input type="checkbox" name="update1" checked="checked" />Please email me updates about
your products.<br/>
<input type="checkbox" name="update2" />Please email me updates about products from thi
rd-party partners.
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="SUBMIT" />
</td></tr>
</table>
</form>
</body>

```

Now, Preview contact_form.php:

Contact ACME Corporation

Name:

Email:

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third-party partners.

SUBMIT

You'll notice here that we've switched the **method** attribute in the HTML form tag from **POST** to **GET**, and we've introduced some PHP echo statements using the **\$_GET[] superglobal**. But so far, no changes have taken place—we still get the same blank form with **contact_form.php** as we did with **contact.html**.

The **htmlspecialchars()** function can be used when obtaining the values for the input tags. For example:

```
<input type="text" size="25" name="email" value="<? echo  
htmlspecialchars($_GET['email'], ENT_QUOTES, 'UTF-8'); ?>" /> </td>
```

Note This function converts some predefined characters to HTML entities and will help to protect your code against cross site scripting. A detailed discussion of web application security is beyond the scope of this course, but please check out the following links for additional information:

[link](#)
[link](#).

Be sure to **Save contact_form.php**, since the **Location:** header will redirect you back to the *saved* version of **contact_form.php**, NOT the Preview version.

Switch to contact.php, and make the following changes in blue:

```
<?php

#We used the superglobal $_GET here
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string;

    header("Location: ".$url);
    exit();

}

extract($_GET, EXTR_PREFIX_SAME, "get");

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWARDED_FOR'];

?>
```

Remember to save contact.php, then switch back to contact_form.php and Preview.

When you submit the form, be sure that you leave one field blank to see what happens:

Contact ACME Corporation

Name:

Email:

Type of Request: I need technical help using the website.

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third-party partners.

Now here's some real progress. When you submit the form with a field or two blank, the form still comes back—but this time, all the fields at the top have been filled in. This is much better, because now the user doesn't have to redo everything.

Let's take another look at the code we used in contact.php:

```
#We used the superglobal $_GET here
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string;
    header("Location: ".$url);
    exit();

}

extract($_GET, EXTR_PREFIX_SAME, "get");
```

We switched our form to have **method=GET** so that our data will come through to our script from the **query string**. The **query string** consists of all the encoded data you see after the question mark (?) in your URL when you submit

the form:

http://josh.onza.net/contact_form.php?name=Trish&email=trish%40myemail.com&whoami=support&subject=Woops%2C+I+left+the+message+field+blank%2C

And, since we have the handy **environment variable** `QUERY_STRING`, we can simply use the `$_SERVER[]` superglobal to grab it and send it back to `contact_form.php`.

And if you look again at `contact_form.php`:

```
<input type="text" size="25" name="name" value="<? echo $_GET['name']; ?>" />
```

You'll see that we were able to harness the **query string** yet again—through the superglobal `$_GET[]`—to fill in the input tags with the customer's original data.

Customizing specific error messages

Now it's time to use our newly-formed script `contact_form.php` to tell the customer exactly what needs to be done. To do this, however, we first need to *manipulate* the **query string** a bit:

In PHP, switch to `contact.php`, and make the following changes, in blue:

```
<?php

#We used the superglobal $_GET here
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixe
    d
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&e
    rror=1";
    header("Location: ".$url);
    exit();

}

extract($_GET, EXTR_PREFIX_SAME, "get");

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name." : ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_
X_FORWARDED_FOR'];

?>
```

Make sure you Save contact.php, and switch to **contact_form.php**:

In PHP, add the following to contact_form.php, in blue:

```
<?php

if ($_GET['error'] == "1") {
    $error_code = 1; //this means that there's been an error and we need to notify the customer
} else {
    $error_code = 0;
}

?>

<body>
<h3>Contact ACME Corporation</h3>
<?
if ($error_code) {
    echo "<div style='color:red'>Please help us with the following:</div>";
}
?>
<form method="GET" action="contact.php">
<table>
<tr>
<td align="right">
Name:
</td>
<td align="left">
<input type="text" size="25" name="name" value="<? echo $_GET['name']; ?>" />
<?
if ($error_code && !($_GET['name'])) {
    echo "<b>Please include your name.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Email:
</td><td align="left">
<input type="text" size="25" name="email" value="<? echo $_GET['email']; ?>" />
<?
if ($error_code && !($_GET['email'])) {
    echo "<b>Please include your email address.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Type of Request:
</td>
<td align="left">
<select name="whoami">
<option value="" />Please choose...
<option value="newcustomer">
if ($_GET['whoami'] == "newcustomer") {
    echo " selected";
}
?> />I am interested in becoming a customer.
<option value="customer">
if ($_GET['whoami'] == "customer") {
    echo " selected";
}
?> />I am a customer with a general question.
<option value="support">
if ($_GET['whoami'] == "support") {
    echo " selected";
}
}
```

```

?> />I need technical help using the website.
<option value="billing"<?
if ($_GET['whoami'] == "billing") {
    echo " selected";
}
?> />I have a billing question.
</select>
<?
if ($error_code && !($_GET['whoami'])) {
    echo "<b>Please choose a request type.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Subject:
</td>
<td align="left">
<input type="text" size="50" max="50" name="subject" value="<? echo $_GET['subject']; ?>" />
<?
if ($error_code && !($_GET['subject'])) {
    echo "<b>Please add a subject for your request.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right" valign="top">
Message:
</td>
<td align="left">
<textarea name="message" cols="50" rows="8">
<? echo $_GET['message']; ?>
</textarea>
<?
if ($error_code && !($_GET['message'])) {
    echo "<b>Please fill in a message for us.</b>";
}
?>
</td>
</tr>
<tr>
<td colspan="2" align="left">
How did you hear about us?
<ul>
<input type="radio" name="found" value="wordofmouth" />Word of Mouth<br/>
<input type="radio" name="found" value="search" />Online Search<br/>
<input type="radio" name="found" value="article" />Printed publication/article<br/>
<input type="radio" name="found" value="website" />Online link/article<br/>
<input type="radio" name="found" value="other" />Other
</ul>
</td>
</tr>
<tr>
<td colspan="2">
<input type="checkbox" name="update1" checked="checked" />Please email me updates about your products.<br/>
<input type="checkbox" name="update2" />Please email me updates about products from third-party partners.
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="SUBMIT" />
</td></tr>

```



```
</table>
</form>
</body>
```

Again, be sure to Save contact_form.php, and then Preview, leaving one field blank. What did you get?

We get something like this:

Contact ACME Corporation

Please help us with the following:

Name: Please include your name.

Email:

Type of Request:

Subject:

Message:

Please fill in

us.

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third party partners.

MUCH better. Now the customer knows exactly what's wrong, he can fix it, and submit the support request easily.

Sending Emails

Finally, we can do what we wanted to do all along: send the support request via email. Never one to let us down, PHP has just the function for us: **mail()**. Let's try it:

In PHP, switch to contact.php, and make the following changes, in blue:

```
<?php

#We used the superglobal $_GET here
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixed
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1"
;
    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}

extract($_GET, EXTR_PREFIX_SAME, "get");

#construct email message
$email_message = "Name: ".$name."
Email: ".$email."
Type of Request: ".$whoami."
Subject: ".$subject."
Message: ".$message."
How you heard about us: ".$found."
User Agent: ".$_SERVER['HTTP_USER_AGENT'].
IP Address: ".$_SERVER['REMOTE_ADDR'];

#construct the email headers
$to = "support@example.com"; //for testing purposes, this should be YOUR email address
.
//We will send the emails from our own server
$from = "anything@yourlogin.oreillystudent.com";

$email_subject = $_GET['subject'];

#now mail
mail($to, $email_subject, $email_message, "From: ".$from);

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";

echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name." ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWA
RDED_FOR'];

?>
```

Save contact.php, switch to `contact_form.php`, Preview, and submit the form.

If you filled in all the required fields, you should get something like before:

Thank You!

Here is a copy of your request:

Name: Trish

Email: trish@myemail.com

Type of Request: support

Subject: Please help!

Message: I can't get the darn thing to work!

How you heard about us: website

update1: on

update2:

You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/4.1 (KHTML, like Gecko) Safari/4.17.9.2

The IP address of the computer you're working on is 63.171.219.74



However, *this* time, if you included your own email in the `$to` variable, you should have a brand new customer support message in your email inbox.

Don't forget to **Save** your work and hand in the **assignments** from your syllabus. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*

Date and Time

You'll find that Date and Time play a huge part in programming - they are useful for timestamps, logs, and are needed in just about every database entry you'll create. And although they're somewhat tricky to harness, PHP has done well in simplifying the process.

Open the two files we were working on before: **contact_form.php** and **contact.php**.

Date and Time Standards

Switch to contact.php, and make the following changes, in green:

```
<?php

#We used the superglobal $_GET here instead of the register globals, for safety
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixed
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1"
;
    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}

extract($_GET, EXTR_PREFIX_SAME, "get");

#construct email message
$email_message = "Name: ".$name."
Email: ".$email."
Type of Request: ".$whoami."
Subject: ".$subject."
Message: ".$message."
How you heard about us: ".$found."
User Agent: ".$_SERVER['HTTP_USER_AGENT']."
IP Address: ".$_SERVER['HTTP_X_FORWARDED_FOR'];

#construct the email headers
$to = "support@example.com"; //for testing purposes, this should be YOUR email address
.
//We will send the emails from our own server
$from = "anything@yourlogin.oreillystudent.com";

$email_subject = "CONTACT #".time().": ".$_GET['subject'];

#now mail
mail($to, $email_subject, $email_message, "From: ".$from);

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #".time().":<br/>";
echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWA
RDED_FOR'];

?>
```

Note

Notice we're breaking one of our own cardinal rules here - doubling up on code that could be taken care of with one function. Feel free to punish us within your own code.

Save `contact.php`, switch to `contact_form.php`, Preview, and submit the form.

If you filled in all the required fields, you should get something like this:

Thank You!

Here is a copy of your request:

CONTACT #1148955473:

Name: Trish

Email: trish@myemail.com

Type of Request: support

Subject: Please help!

Message: I can't get the darn thing to work!

How you heard about us: website

update1: on

update2:

You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/ (KHTML, like Gecko) Safari/417.9.2

The IP address of the computer you're working on is 63.171.219.74

Although we're using it as a **timestamp** here, the number we got actually measures how many seconds have passed since **Unix Epoch** -- that's a fancy name for January 1st, 1970, at midnight (00:00:00) GMT. Why is that time the **Unix Epoch**? No good reason really, except that some early computer scientists agreed on it a long time ago as a **date and time standard**.

Sounds nerdy, but it's really a good thing - it enables us to harness date and time, not only in PHP, but also in MySQL and lots of other technology languages. For instance, you'll be using PHP functions to process SQL timestamps in later courses.

Date and Time Functions

Obviously, **date and time standards** weren't created for us to use merely as a unique identification number - although that's handy. What else can they do for us? Enter the built-in PHP functions.

Switch to contact.php, and make the following changes, in green:

```
<?php

#We used the superglobal $_GET here instead of the register globals, for safety
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixed
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1"
;
    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}

extract($_GET, EXTR_PREFIX_SAME, "get");

#construct email message
$email_message = "Message Date: ".date("F d, Y h:i a")."
Name: ".$name."
Email: ".$email."
Type of Request: ".$whoami."
Subject: ".$subject."
Message: ".$message."
How you heard about us: ".$found."
User Agent: ".$_SERVER['HTTP_USER_AGENT'].
IP Address: ".$_SERVER['HTTP_X_FORWARDED_FOR'];

#construct the email headers
$to = "support@example.com"; //for testing purposes, this should be YOUR email address
.
//We will send the emails from our own server
$from = "anything@yourlogin.oreillystudent.com";

$email_subject = "CONTACT #".time().": ".$_GET['subject'];

#now mail
mail($to, $email_subject, $email_message, "From: ".$from);

echo "<h3>Thank you!</h3>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #".time().":<br/>";
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name." : ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWA
RDED_FOR'];
```

?>

Again, if you **Save contact.php**, then Preview contact_form.php, you might get something like this:

Thank You!

Here is a copy of your request:

CONTACT #1148955473:

Message Date: May 29, 2006 10:25 pm

Name: Trish

Email: trish@myemail.com

Type of Request: support

Subject: Please help!

Message: I can't get the darn thing to work!

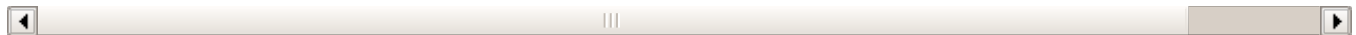
How you heard about us: website

update1: on

update2:

You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/4.1 (KHTML, like Gecko) Safari/417.9.2

The IP address of the computer you're working on is 63.171.219.74



This time, instead of a cryptic timestamp, the date of the message has been nicely formatted for us through the **date()** function.

Let's take another look:

```
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
```

The **parameter** for the **date()** function is a special coded **format** that PHP replaces with the proper time/date data. For instance, **"F"** is replaced with the name of the month - in our case, **May**, and **"a"** is replaced with **am or pm**, depending on the time - in our case, **pm**. [Here](#) is the php.net reference for time/date formats.

format character	Description	Example returned values
Day	---	---
<i>d</i>	Day of the month, 2 digits with leading zeros	01 to 31
<i>D</i>	A textual representation of a day, three letters	Mon through Sun
<i>j</i>	Day of the month without leading zeros	1 to 31
<i>l</i> (lowercase 'L')	A full textual representation of the day of the week	Sunday through Saturday
<i>N</i>	ISO-8601 numeric representation of the day of the week (added in PHP 5.1.0)	1 (for Monday) through 7 (for Sunday)
<i>S</i>	English ordinal suffix for the day of the month, 2 characters	st, nd, rd or th. Works well with <i>j</i>
<i>w</i>	Numeric representation of the day of the week	0 (for Sunday) through 6 (for Saturday)
<i>z</i>	The day of the year (starting from 0)	0 through 365
Week	---	---

<i>W</i>	ISO-8601 week number of year, weeks starting on Monday (added in PHP 4.1.0)	Example: 42 (the 42nd week in the year)
Month	---	---
<i>F</i>	A full textual representation of a month, such as January or March	<i>January</i> through <i>December</i>
<i>m</i>	Numeric representation of a month, with leading zeros	01 through 12
<i>M</i>	A short textual representation of a month, three letters	<i>Jan</i> through <i>Dec</i>
<i>n</i>	Numeric representation of a month, without leading zeros	1 through 12
<i>t</i>	Number of days in the given month	28 through 31
Year	---	---
<i>L</i>	Whether it's a leap year	1 if it is a leap year, 0 otherwise.
<i>o</i>	ISO-8601 year number. This has the same value as <i>Y</i> , except that if the ISO week number (<i>W</i>) belongs to the previous or next year, that year is used instead. (added in PHP 5.1.0)	Examples: 1999 or 2003
<i>Y</i>	A full numeric representation of a year, 4 digits	Examples: 1999 or 2003
<i>y</i>	A two digit representation of a year	Examples: 99 or 03
Time	---	---
<i>a</i>	Lowercase Ante meridiem and Post meridiem	<i>am</i> or <i>pm</i>
<i>A</i>	Uppercase Ante meridiem and Post meridiem	<i>AM</i> or <i>PM</i>
<i>B</i>	Swatch Internet time	000 through 999
<i>g</i>	12-hour format of an hour without leading zeros	1 through 12
<i>G</i>	24-hour format of an hour without leading zeros	0 through 23
<i>h</i>	12-hour format of an hour with leading zeros	01 through 12
<i>H</i>	24-hour format of an hour with leading zeros	00 through 23
<i>i</i>	Minutes with leading zeros	00 to 59
<i>s</i>	Seconds, with leading zeros	00 through 59
Timezone	---	---
<i>e</i>	Timezone identifier (added in PHP 5.1.0)	Examples: <i>UTC</i> , <i>GMT</i> , <i>Atlantic/Azores</i>
<i>I</i> (capital i)	Whether or not the date is in daylight savings time	1 if Daylight Savings Time, 0 otherwise.
<i>O</i>	Difference to Greenwich time (GMT) in hours	Example: +0200
<i>P</i>	Difference to Greenwich time (GMT) with colon between hours and minutes (added in PHP 5.1.3)	Example: +02:00
<i>T</i>	Timezone setting of this machine	Examples: <i>EST</i> , <i>MDT</i> ...
<i>Z</i>	Timezone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.	-43200 through 43200
Full Date/Time	---	---
<i>c</i>	ISO 8601 date (added in PHP 5)	2004-02-12T15:19:21+00:00
<i>r</i>	RFC 2822 formatted date	Example: <i>Thu, 21 Dec 2000 16:01:07 +0200</i>

<i>U</i>	Seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)
----------	--

Constructing Dates and Times

Now, suppose Acme, Inc. had a customer service policy claiming "We'll get back to you in 48 hours." You'll want to use the date of the message to give the customer support representative an idea of the deadline she has.

Make sure you have contact.php, and make the following changes, in green:

```
<?php

#We used the superglobal $_GET here instead of the register globals, for safety
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixe
    d
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&e
    rror=1";
    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}

extract($_GET, EXTR_PREFIX_SAME, "get");

#construct email message

#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;
$deadline_str = $deadline_array['month']." ".$deadline_day." ".$deadline_array['
year'];

$email_message = "Message Date: ".date("F d, Y h:i a")."
Please reply by: ".$deadline_str."
Name: ".$name."
Email: ".$email."
Type of Request: ".$whoami."
Subject: ".$subject."
Message: ".$message."
How you heard about us: ".$found."
User Agent: ".$_SERVER['HTTP_USER_AGENT']."
IP Address: ".$_SERVER['HTTP_X_FORWARDED_FOR'];

#construct the email headers
$to = "support@example.com"; //for testing purposes, this should be YOUR email
address.
//We will send the emails from our own server
$from = "anything@yourlogin.oreillystudent.com";

$email_subject = "CONTACT #".time().": ".$_GET['subject'];

#now mail
mail($to, $email_subject, $email_message, "From: ".$from);

echo "<h3>Thank you!</h3>";
echo "We'll get back to you by ".$deadline_str."<br/>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #".time().":<br/>";
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
```

```

        $element_name = "update".$i;
        echo $element_name."": ";
        echo $$element_name;
        echo "<br/>";
    }

    echo "You are currently working on " . $_SERVER['HTTP_USER_AGENT'];
    echo "<br/>The IP address of the computer you're working on is " . $_SERVER['HTTP_X_FORWARDED_FOR'];

?>

```

Save contact.php, switch to contact_form.php and Preview:

Thank You!

We'll get back to you by May 31 2006.
Here is a copy of your request:

CONTACT #1148955473:
 Message Date: May 29, 2006 10:25 pm
 Name: Trish
 Email: trish@myemail.com
 Type of Request: support
 Subject: Please help!
 Message: I can't get the darn thing to work!
 How you heard about us: website
 update1: on
 update2:
 You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/417.9.2
 The IP address of the computer you're working on is 63.171.219.74



Take another look:

```

#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;
$deadline_str = $deadline_array['month'] . " " . $deadline_day . " " . $deadline_array['year'];

```

Here, the function **getdate()**, like **time()**, gets a stamp of the current time. However, instead of just an integer, **getdate()** extracts the data and outputs an **associative array** that looks a bit like this:

Here's what a `getdate()` output array might look like:

```
Array
(
    [seconds] => 40
    [minutes] => 58
    [hours]   => 21
    [mday]    => 29
    [wday]    => 1
    [mon]     => 5
    [year]    => 2006
    [yday]    => 160
    [weekday] => Monday
    [month]   => May
    [0]       => 1055901520
)
```

This array makes it easy to construct a new date relative to the current date - all we have to do is add 2 to the **'mday'** array value, and suddenly we have a deadline for the customer support representative. Fast service means happy customers.

But wait a minute - what if today was, say, the 31st of May? Just adding 2 to that will give you an invalid date. We could do a series of **if** statements to fix this, but that's a lot of unwieldy code. Luckily, PHP has yet another handy function to help us.

In PHP, try adding the following green code to contact.php:

```
<?php

#We used the superglobal $_GET here instead of the register globals, for safety
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixe
    d
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&e
    rror=1";
    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}

extract($_GET, EXTR_PREFIX_SAME, "get");

#construct email message
#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;

$deadline_stamp = mktime($deadline_array['hours'],$deadline_array['minutes'],$de
adline_array['seconds'],
    $deadline_array['mon'],$deadline_day,$deadline_array['year']);
$deadline_str = date("F d, Y", $deadline_stamp);

$email_message = "Message Date: ".date("F d, Y h:i a")."
    Please reply by: ".$deadline_str."
    Name: ".$name."
    Email: ".$email."
    Type of Request: ".$whoami."
    Subject: ".$subject."
    Message: ".$message."
    How you heard about us: ".$found."
    User Agent: ".$_SERVER['HTTP_USER_AGENT']."
    IP Address: ".$_SERVER['HTTP_X_FORWARDED_FOR'];

#construct the email headers
$to = "support@example.com"; //for testing purposes, this should be YOUR email
address.
//We will send the emails from our own server
$from = "anything@yourlogin.oreillystudent.com";

$email_subject = "CONTACT #" . time() . ": ".$_GET['subject'];

#now mail
mail($to, $email_subject, $email_message, "From: ".$from);

echo "<h3>Thank you!</h3>";
echo "We'll get back to you by ".$deadline_str."<br/>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #" . time() . "<br/>";
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";
```

```

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on "._SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is "._SERVER['HTTP_
X_FORWARDED_FOR'];

?>

```

Save contact.php, switch to contact_form.php and Preview:

Thank You!

We'll get back to you by June 2 2006.
Here is a copy of your request:

CONTACT #1148962509:
 Message Date: May 31, 2006 12:20 pm
 Name: Trish
 Email: trish@myemail.com
 Type of Request: support
 Subject: Please help!
 Message: I can't get the darn thing to work!
 How you heard about us: website
 update1: on
 update2:
 You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/417.9.2 (KHTML, like Gecko) Safari/417.9.2
 The IP address of the computer you're working on is 63.171.219.74



Take one more look:

```

$deadline_stamp = mktime($deadline_array['hours'],$deadline_array['minutes'],$de
adline_array['seconds'],
    $deadline_array['mon'],$deadline_day,$deadline_array['year']);
$deadline_str = date("F d, Y", $deadline_stamp);

```

The function **mktime()** fixes all those pesky date problems. It takes in the parameter data of the date you want to format, and creates the original **timestamp**, which we then plug into **date()** to format properly. Problem solved!

Don't forget to **Save** your work and hand in your **assignments** from your syllabus. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Using Files

So far, we've created a simple corporate contact form that accounts for user error and friendly reminders, successful output, and sending of the proper message. Pretty robust, but at the same time, you have to admit, it's pretty ugly. And it's not the most professional-looking interface on the web either.

In a perfect world, we'd have lots of time to keep tweaking the PHP script to make every page look just so. But in most situations, you won't have the luxury of extra time, and you probably won't even be allowed to dictate how the page looks. Not when there's a graphic designer down the hall. Just the same, you don't want the graphic designer down the hall messing with your PHP scripts either. Here's where **file templates** come in real handy.

Including and Requiring Files

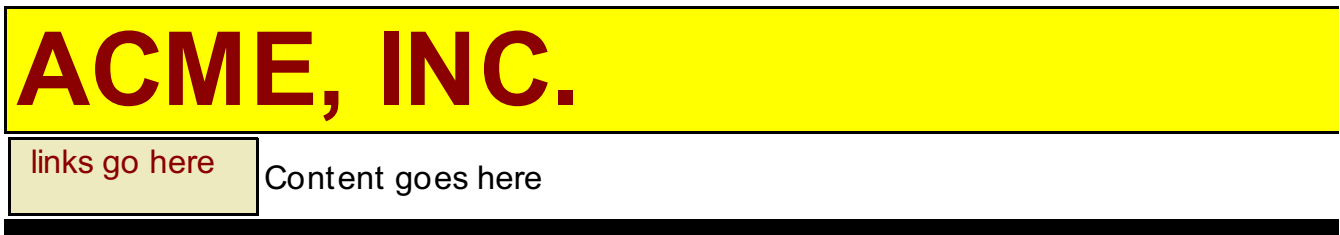
Fire up CodeRunner and open up the two files we were working on before: **contact_form.php** and **contact.php**. After you do this, switch CodeRunner to **HTML syntax**. For just a moment, we're going to pretend that we are the graphic designers down the hall.

In HTML, type the following, in blue:

```
<html>
<head>
<title>Acme, Inc.</title>
<link rel="stylesheet" href="http://students.oreillyschool.com/resource/php_lesson.css"
  type="text/css" />
</head>
<body>
<div class="topbar">
ACME, INC.
</div>
<table>
<tr><td class="sidebar" valign="top">
links go here
</td><td class="content">
Content goes here

</td></tr></table>
<div class="bottombar">
</div>
</body>
</html>
```

Preview this:



What we have here is a basic "C-Clamp" design template for a corporate web page: logo on top, links on the side, something on the bottom to wrap the content nicely, and a CSS file to add a little style (here we've provided one for you). This will make our contact form look slightly better than it did before.

But how do you most easily place our content within this C-Clamp? You could simply *embed* the HTML into the PHP script itself, but this creates a big problem - if the graphic designer decides to make a change, you're stuck making that

same change in every PHP script you've written. And if you work for a large corporation, this could mean dozens, even hundreds of files.

It would be great if you could *reuse* the code, like when you create PHP functions.

In HTML, remove the second half of our C-Clamp:

```
<html>
<head>
<title>Acme, Inc.</title>
<link rel="stylesheet" href="http://students.oreillyschool.com/resource/php_lesson.css"
  type="text/css" />
</head>
<body>
<div class="topbar">
ACME, INC.
</div>
<table>
<tr><td class="sidebar" valign=top>
links go here
</td><td class="content">
```

Now, **Save** this file and name it **template_top.inc**.

Note Why use **.inc**? Just for clarity - this isn't a complete HTML file, so no need to name it with a **.html** suffix.

In HTML, create a NEW file, containing the second half of our C-Clamp:

```
</td></tr></table>
<div class="bottombar">
</div>
</body>
</html>
```

Save this file and name it **template_bottom.inc**.

Add the following to contact_form.php, in green:

```
<?php

    require($_SERVER['DOCUMENT_ROOT']. "/template_top.inc");

if ($_GET['error'] == "1") {
    $error_code = 1; //this means that there's been an error and we need to notify the
customer
}

?>
<body>
<h3>Contact ACME Corporation</h3>
<?
if ($error_code) {
    echo "<font color=red>Please help us with the following:</font>";
}
?>
<form method=GET action="contact.php">
<table>
<tr>
<td align="right">
Name:
</td>
<td align="left">
<input type="text" size="25" name="name" value="<? echo $_GET['name']; ?>">
<?
if ($error_code && !($_GET['name'])) {
    echo "<b>Please include your name.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Email:
</td><td align="left">
<input type="text" size="25" name="email" value="<? echo $_GET['email']; ?>">
<?
if ($error_code && !($_GET['email'])) {
    echo "<b>Please include your email address.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Type of Request:
</td>
<td align="left">
<select name="whoami">
<option value="">Please choose...
<option value="newcustomer">
<?
if ($_GET['whoami'] == "newcustomer") {
    echo " selected";
}
?>>I am interested in becoming a customer.
<option value="customer">
<?
if ($_GET['whoami'] == "customer") {
    echo " selected";
}
?>>I am a customer with a general question.
<option value="support">
<?
if ($_GET['whoami'] == "support") {
    echo " selected";
}
```

```

}
?>>I need technical help using the website.
<option value="billing"<?
if ($_GET['whoami'] == "billing") {
    echo " selected";
}
?>>I have a billing question.
</select>
<?
if ($error_code && !($_GET['whoami'])) {
    echo "<b>Please choose a request type.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Subject:
</td>
<td align="left">
<input type="text" size="50" max="50" name="subject" value="<? echo $_GET['subject']; ?
>">
<?
if ($error_code && !($_GET['subject'])) {
    echo "<b>Please add a subject for your request.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right" valign="top">
Message:
</td>
<td align="left">
<textarea name="message" cols=50 rows=8>
<? echo $_GET['message']; ?>
</textarea>
<?
if ($error_code && !($_GET['message'])) {
    echo "<b>Please fill in a message for us.</b>";
}
?>
</td>
</tr>
<tr>
<td colspan="2" align="left">
How did you hear about us?
<ul>
<input type="radio" name="found" value="wordofmouth">Word of Mouth<br>
<input type="radio" name="found" value="search">Online Search<br>
<input type="radio" name="found" value="article">Printed publication/article<br>
<input type="radio" name="found" value="website">Online link/article<br>
<input type="radio" name="found" value="other">Other
</ul>
</td>
</tr>
<tr>
<td colspan="2">
<input type="checkbox" name="update1" checked>Please email me updates about your produc
ts.<br>
<input type="checkbox" name="update2">Please email me updates about products from third
-party partners.
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="SUBMIT">
</td></tr>

```

```

</table>
</form>

<?
    require($_SERVER['DOCUMENT_ROOT']. "/template_bottom.inc");
?>

</body>

```

Be sure and **Save** `contact_form.php`.

Now PREVIEW:

ACME, INC.

links go here

Contact ACME Corporation

Name:

Email:

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

☐ Word of Mouth
☐ Online Search
☐ Printed publication/article
☐ Online link/article
☐ Other

☒ Please email me updates about your products.
☐ Please email me updates about products from third-party partners.

SUBMIT

Even with the extreme simplicity of our C-Clamp template, this looks much better than it did before.

Take another look at the code:

```

require($_SERVER['DOCUMENT_ROOT']. "/template_top.inc");

```

The PHP built-in function **require()** takes a filename as its **parameter**, and imports all the data from that filename into that exact place within the PHP code. It's as if you had written the code right in.

We can do this with **contact.php** as well.

Switch to contact.php and add the following green code:

```
<?php

#Remember, if you place any output before a header() call, you'll get an error.
#We used the superglobal $_GET here instead of the register globals, for safety
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixed
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1"
;
    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}
extract($_GET, EXTR_PREFIX_SAME, "get");
#construct email message
#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;

$deadline_stamp = mktime($deadline_array['hours'],$deadline_array['minutes'],$deadline_
array['seconds'],
    $deadline_array['mon'],$deadline_day,$deadline_array['year']);
$deadline_str = date("F d, Y", $deadline_stamp);

$email_message = "Message Date: ".date("F d, Y h:i a")."<br>
    Please reply by: ".$deadline_str."<br>
    Name: ".$name."<br>
    Email: ".$email."<br>
    Type of Request: ".$whoami."<br>
    Subject: ".$subject."<br>
    Message: ".$message."<br>
    How you heard about us: ".$found."<br>
    User Agent: ".$_SERVER['HTTP_USER_AGENT']."<br>
    IP Address: ".$_SERVER['HTTP_X_FORWARDED_FOR'];

#construct the email headers
$to = "support@example.com"; //for testing purposes, this should be YOUR email address
.
//We will send the emails from our own server
$from = "anything@yourlogin.oreillystudent.com";

$email_subject = "CONTACT #".time().": ".$_GET['subject'];

$headers = "From: " . $from . "\r\n";
$headers .= "MIME-Version: 1.0" . "\n"; //these headers will allow our HTML tags to be
    displayed in the email
$headers .= "Content-type: text/html; charset=iso-8859-1" . "\r\n";

#now mail
mail($to, $email_subject, $email_message, $headers);

include($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

echo "<h3>Thank you!</h3>";
echo "We'll get back to you by ".$deadline_str."<br>";
echo "Here is a copy of your request:<br><br>";
echo "CONTACT #".time().":<br>";
echo "Message Date: ".date("F d, Y h:i a")."<br>";
```

```

echo "Name: ".$name."<br>";
echo "Email: ".$email."<br>";
echo "Type of Request: ".$whoami."<br>";
echo "Subject: ".$subject."<br>";
echo "Message: ".$message."<br>";
echo "How you heard about us: ".$found."<br>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWARDED_FOR'];

include($_SERVER['DOCUMENT_ROOT']."/template_bottom.inc");

?>

```

Save contact.php. Now when you view **contact_form.php** and submit the form, you should see something like this:

ACME, INC.

links go here

Thank you!

We'll get back to you by June 07, 2006.
Here is a copy of your request:

CONTACT #1149489921:

Message Date: June 05, 2006 01:45 pm

Name: Trish

Email: trish@myemail.com

Type of Request: support

Subject: Please help!

Message: I can't get the darn thing to work!

How you heard about us: other

update1: on

update2:

You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; er AppleWebKit/4.18 (KHTML, like Gecko) Safari/4.17.9.2

The IP address of the computer you're working on is 63.171.219.74



Note

This time, instead of **require()** we used **include()**. What's the difference? If for some reason the URL doesn't exist, **require()** will give you a PHP error, whereas **include()** will just skip that URL.

Reading and Writing Files

Now that we've got the web interface looking better, let's work on the email. In this case, there's no clear-cut beginning and ending template, rather, the data is peppered throughout the email. So if we want to use a template with this, we'll have to find a way to insert the data into the template, instead of the other way around.

First, let's see how we want the template to look. Switch to **HTML**, and create a **text-only file** that looks something like below.

Make sure you're in HTML, and type the following into a new file:

You have just received a customer email. Please respond to this email by #DEADLINE#.
Details are below:

```
<table>
<tr><td width="100" align="right">Message Type: </td><td>#WHOAMI#</td></tr>
<tr><td width="100" align="right">Message Date: </td><td>#DATE#</td></tr>
<tr><td width="100" align="right">Name: </td><td>#NAME#</td></tr>
<tr><td width="100" align="right">Email: </td><td>#EMAIL#</td></tr>
<tr><td width="100" align="right">IP Address: </td><td>#IP#</td></tr>
<tr><td width="100" align="right">Platform: </td><td>#AGENT#</td></tr>
</table>
```

```
<b>Subject: #SUBJECT#
<br>
#MESSAGE#</b>
<br><br>
This customer found us through #FOUND#.<br>
#CONTACT#
```

Save this text file, and call it **email_template.txt**. Now let's go back to **contact.php**.

Switch to contact.php and make the following changes, in green:

```
<?php
```

```
function mail_message($data_array, $template_file, $deadline_str) {

    #get template contents, and replace variables with data
    $email_message = file_get_contents($template_file);
    $email_message = str_replace("#DEADLINE#", $deadline_str, $email_message);
    $email_message = str_replace("#WHOAMI#", $data_array['whoami'], $email_message);
    $email_message = str_replace("#DATE#", date("F d, Y h:i a"), $email_message);
    $email_message = str_replace("#NAME#", $data_array['name'], $email_message);
    $email_message = str_replace("#EMAIL#", $data_array['email'], $email_message);
    $email_message = str_replace("#IP#", $_SERVER['HTTP_X_FORWARDED_FOR'], $email_message);
    $email_message = str_replace("#AGENT#", $_SERVER['HTTP_USER_AGENT'], $email_message);

    $email_message = str_replace("#SUBJECT#", $data_array['subject'], $email_message);
    $email_message = str_replace("#MESSAGE#", $data_array['message'], $email_message);
    $email_message = str_replace("#FOUND#", $data_array['found'], $email_message);

    #include whether or not to contact the customer with offers in the future
    $contact = "";
    if (isset($data_array['update1'])) {
        $contact = $contact." Please email updates about your products.<br/>";
    }
    if (isset($data_array['update2'])) {
        $contact = $contact." Please email updates about products from third-party partners.<br/>";
    }
    $email_message = str_replace("#CONTACT#", $contact, $email_message);

    #construct the email headers
    $to = "support@example.com"; //for testing purposes, this should be YOUR email address.
    $from = $data_array['email'];
    $email_subject = "CONTACT #".time().": ".$data_array['subject'];

    $headers = "From: " . $from . "\r\n";
    $headers .= 'MIME-Version: 1.0' . "\n"; //these headers will allow our HTML tags to be displayed in the email
    $headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";

    #now mail
    mail($to, $email_subject, $email_message, $headers);
}

#Remember, if you place any output before a header() call, you'll get an error.
#We used the superglobal $_GET here instead of the register globals, for safety
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami'] && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixed
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1";

    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}
```

```

}

extract($_GET, EXTR_PREFIX_SAME, "get");

#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;

$deadline_stamp = mktime($deadline_array['hours'],$deadline_array['minutes'],$deadli
ne_array['seconds'],
    $deadline_array['mon'],$deadline_day,$deadline_array['year']);
$deadline_str = date("F d, Y", $deadline_stamp);

//DOCUMENT_ROOT is the file path leading up to the template name.
mail_message($_GET, $_SERVER['DOCUMENT_ROOT']."/email_template.txt", $deadline_str);

include($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

echo "<h3>Thank you!</h3>";
echo "We'll get back to you by ".$deadline_str."<br/>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #".time().":<br/>";
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWA
RDED_FOR'];

include($_SERVER['DOCUMENT_ROOT']."/template_bottom.inc");

?>

```

Save contact.php, then view and submit the form in **contact_form.php**. If you used your own email address as the **\$to** variable, you should have received an email in your INBOX like before. However, this time it should look a little better.

You should have received an email like this:

Date: Thu, 8 Jun 2006 17:03:11 -0500
From: trish@myemail.com
To: support@acmeinc.com
Subject: CONTACT #1149804191: Please help!

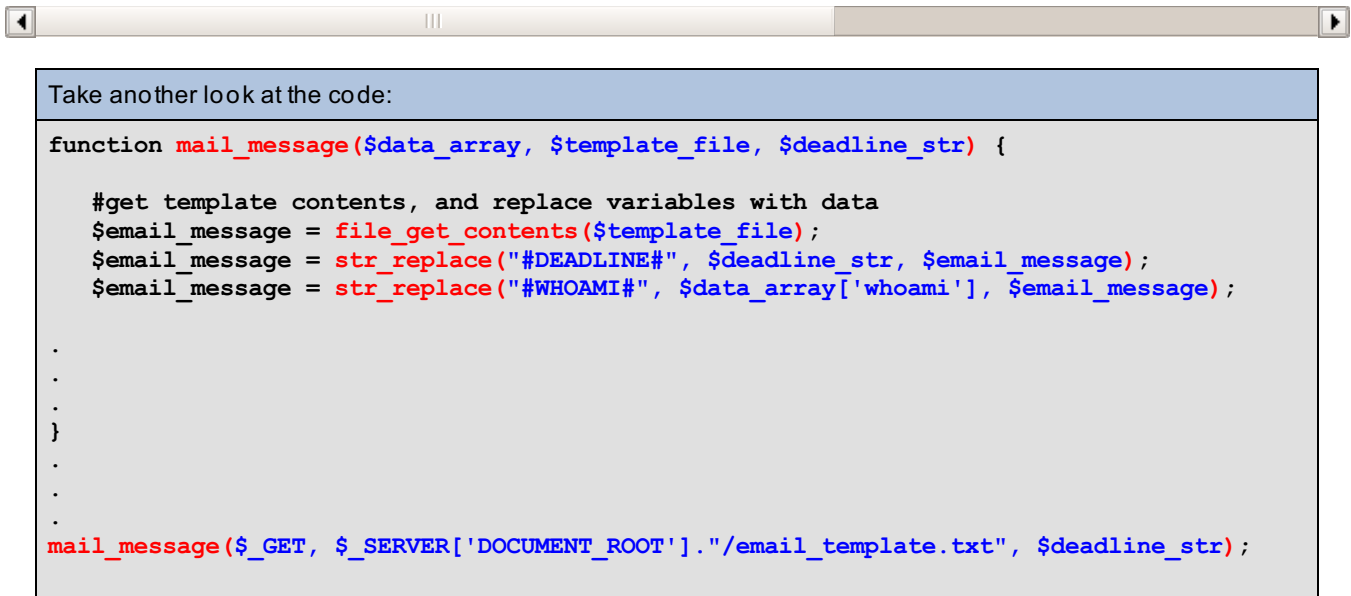
You have just received a customer email. Please respond to this email by June 10,
Details are below:

Message Type: support
Message Date: June 08, 2006 05:03 pm
Name: Trish
Email: trish@myemail.com
IP Address: 12.149.132.162
Platform: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit

Subject: Please help!

I can't get the darn thing to work!

This customer found us through wordofmouth.
Please email updates about your products.



```
function mail_message($data_array, $template_file, $deadline_str) {  
  
    #get template contents, and replace variables with data  
    $email_message = file_get_contents($template_file);  
    $email_message = str_replace("#DEADLINE#", $deadline_str, $email_message);  
    $email_message = str_replace("#WHOAMI#", $data_array['whoami'], $email_message);  
  
    .  
    .  
    .  
}  
.  
.  
.  
  
mail_message($_GET, $_SERVER['DOCUMENT_ROOT']. "/email_template.txt", $deadline_str);
```

Here, we created a function called **mail_message()**, which takes three parameters -- **\$data_array**, **\$template_file**, and **\$deadline_str**. **\$data_array** contains all the form data, because we pass the **\$_GET** superglobal array into it. **\$template_file** is the full path to the template file we want to use - in our case, we passed in the path to "email_template.txt" that we created earlier. And **\$deadline_str** is the formatted string of the date by which we want the message answered.

We used the built-in PHP function **file_get_contents()** to import our email template file into a string, **\$email_message**. Then, one by one, we replace each of our template variables with the corresponding form data, using the built-in function **str_replace()**. Go to php.net to read more about [file_get_contents\(\)](#) or [str_replace\(\)](#).

By making the support email easier to read -- and obtaining as much user information as possible -- you've improved efficiency in Acme's customer support process. Go ahead, demand a raise. You deserve it.

Allowing Users to Download Files

To make things a little more interesting, it turns out that Acme wants every customer who sends in a support email to

be allowed to download its informational brochure, a PDF document.

Now, technically you could just include a link to the PDF document itself, if the document is in a web-accessible directory. However, most of the time corporations don't want their downloadable files to be in a public area for anyone and everyone to download. This is especially true when electronic documents are for purchase, like marketing reports or copyrighted materials.

In your case, we've placed the brochure, called **acme_brochure.pdf**, in a hidden directory called **.php_files/** within your account. You can't view this file through the web, but you need to allow web users of your choosing to download it. What do you do?

In PHP, create a new file, called download.php:

```
<?php

$filepath = $_SERVER['DOCUMENT_ROOT']. "/.php_files/acme_brochure.pdf";
if (file_exists($filepath)) {
    header("Content-Type: application/force-download");
    header("Content-Disposition: filename=\"brochure.pdf\"");
    $fd = fopen($filepath, 'rb');
    fpassthru($fd);
    fclose($fd);
}

?>
```

Save download.php.

Now, switch back to contact.php and make the following changes, in green and blue:

```
<?php

function mail_message($data_array, $template_file, $deadline_str) {

    #get template contents, and replace variables with data
    $email_message = file_get_contents($template_file);
    $email_message = str_replace("#DEADLINE#", $deadline_str, $email_message);
    $email_message = str_replace("#WHOAMI#", $data_array['whoami'], $email_message);
    $email_message = str_replace("#DATE#", date("F d, Y h:i a"), $email_message);
    $email_message = str_replace("#NAME#", $data_array['name'], $email_message);
    $email_message = str_replace("#EMAIL#", $data_array['email'], $email_message);
    $email_message = str_replace("#IP#", $_SERVER['HTTP_X_FORWARDED_FOR'], $email_message);
    $email_message = str_replace("#AGENT#", $_SERVER['HTTP_USER_AGENT'], $email_message);

    $email_message = str_replace("#SUBJECT#", $data_array['subject'], $email_message);
    $email_message = str_replace("#MESSAGE#", $data_array['message'], $email_message);
    $email_message = str_replace("#FOUND#", $data_array['found'], $email_message);

    #include whether or not to contact the customer with offers in the future
    $contact = "";
    if (isset($data_array['update1'])) {
        $contact = $contact." Please email updates about your products.<br/>";
    }
    if (isset($data_array['update2'])) {
        $contact = $contact." Please email updates about products from third-party partners.<br/>";
    }
    $email_message = str_replace("#CONTACT#", $contact, $email_message);

    #construct the email headers
    $to = "support@example.com"; //for testing purposes, this should be YOUR email address.
    $from = $data_array['email'];
    $email_subject = "CONTACT #" . time() . ": " . $data_array['subject'];

    $headers = "From: " . $from . "\r\n";
    $headers .= 'MIME-Version: 1.0' . "\n"; //these headers will allow our HTML tags to be displayed in the email
    $headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";

    #now mail
    mail($to, $email_subject, $email_message, $headers);
}

#Remember, if you place any output before a header() call, you'll get an error.
#We used the superglobal $_GET here instead of the register globals, for safety
if (!($_GET['name'] && $_GET['email'] && $_GET['whoami'] && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixed
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1";

    header("Location: ".$url);
    exit(); //stop the rest of the program from happening
}

extract($_GET, EXTR_PREFIX_SAME, "get");
```

```

#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;

$deadline_stamp = mktime($deadline_array['hours'],$deadline_array['minutes'],$deadli
ne_array['seconds'],
    $deadline_array['mon'],$deadline_day,$deadline_array['year']);
$deadline_str = date("F d, Y", $deadline_stamp);

//DOCUMENT_ROOT is the file path leading up to the template name.
mail_message($_GET, $_SERVER['DOCUMENT_ROOT']."/email_template.txt", $deadline_str);

include($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

echo "<h3>Thank you!</h3>";
echo "We'll get back to you by ".$deadline_str."<br/>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #".time().":<br/>";
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
echo "Name: ".$name."<br/>";
echo "Email: ".$email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWA
RDED_FOR'];

?>
<br/><br/><a href="download.php"><b>Download our PDF brochure!</b></a>
<?

include($_SERVER['DOCUMENT_ROOT']."/template_bottom.inc");

?>

```

Save **contact.php**, then view **contact_form.php** and submit the form:

It should look something like this:

ACME, INC.

links go here

Thank you!

We'll get back to you by June 10, 2006.

Here is a copy of your request:

CONTACT #1149809625:

Message Date: June 08, 2006 06:33 pm

Name: Trish

Email: trish@myemail.com

Type of Request: support

Subject: Please Help!

Message: I can't get the darn thing to work!

How you heard about us: wordofmouth

update1: on

update2:

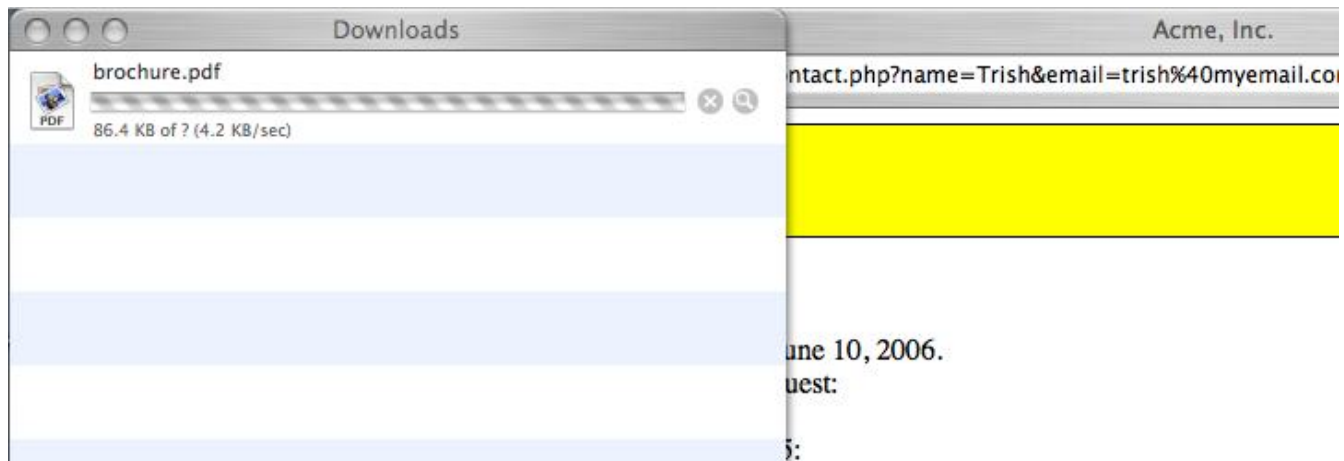
You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X; er
AppleWebKit/4.18 (KHTML, like Gecko) Safari/4.17.9.2

The IP address of the computer you're working on is 63.171.219.74

[Download our PDF brochure!](#)



Now click the link. Did the PDF file download to your computer? You may have seen something like this:



How were we able to do that? Take another look at the code in download.php:

OBSERVE:

```
$filepath = $_SERVER['DOCUMENT_ROOT'].'/.php_files/acme_brochure.pdf';
if (file_exists($filepath)) {
    header("Content-Type: application/force-download");
    header("Content-Disposition: filename=\"brochure.pdf\"");
    $fd = fopen($filepath, 'rb');
    fpassthru($fd);
    fclose($fd);
}
```

First, the built-in function `file_exists()` does exactly what it says - it returns TRUE or FALSE based upon the existence of the parameter `$filepath`, which we set to the path of Acme's hidden PDF brochure in our account. Since it does exist, we use `header()` to output two **HTTP headers**. The header "**Content-Type**" is extremely important, as it tells the web browser that we are preparing to download data that is NOT in an HTML or text format, but in fact an application. Find out what happens if you leave this header out. The header "**Content-Disposition**" is optional, but we used it to create a generic name for the downloaded file.

Note

In the case of PDF files, you can also use the header "**Content-Type: application/pdf**". What's the difference? Some browsers allow PDF files to be opened within the browser itself, without having to download them to the computer's hard drive. Try it out and see what happens in your own browser.

Then, the built-in function `fopen()` creates a **file stream** pointing to our `acme_brochure.pdf` file, and *binds* it to the handle `$fd`. The parameter `'rb'` specifies that the file should be opened in **read-only, binary mode** -- binary, again, because it's not a text file. `fpassthru()` then sends all the file data through to the **output buffer** -- and because we specified through `header()` what the browser should do with that output, this launches your computer's download manager. `fclose()` simply closes the **file stream** `$fd`, to clean things up.

Don't forget to **Save** your work and hand in your **assignments** from your syllabus. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Cookies and Sessions

Learning about **cookies** and **sessions** is essential for programming PHP in the 21st century. You see, web users just aren't as patient as they used to be - they want websites that are incredibly easy for them to use and reuse, without having to repeat themselves over and over again. And their attention spans are shorter as well, meaning corporate websites in particular must compete by targeting the user as specifically as possible.

"*Know Thy User*", as they say. But how?

Using Cookies



Mmmm, cookies. Well, no, not *those* kinds of cookies. Although we would certainly revisit a web site for free cookies any day, unfortunately, downloading chocolate-chip goodness just hasn't been invented yet. *Sigh...*

Okay, so what are **browser cookies**? Let's find out. Fire up CodeRunner in **PHP**, and **open** your files **contact_form.php** and **contact.php**.

Add the following to contact_form.php, in green and blue:

```
<?php

require($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

if ($_GET['error'] == "1") {
    $error_code = 1; //this means that there's been an error and we need to notify the
customer
}

?>

<h3>Contact ACME Corporation</h3>
<?
if ($error_code) {
echo "<div style='color:red'>Please help us with the following:</div>";
}
?>
<form method="GET" action="contact.php">
<table>
<tr>
<td align="right">
Name:
</td>
<td align="left">
<?
if ($_COOKIE['name']) {
    echo $_COOKIE['name'];
}
else {
?>
<input type="text" size="25" name="name" value="<? echo $_GET['name']; ?>" />
<input type="checkbox" name="remember" /> Remember me on this computer
<?
}
if ($error_code && !($_GET['name'] || $_COOKIE['name'])) {
    echo "<b>Please include your name.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Email:
</td><td align="left">
<?
if ($_COOKIE['email']) {
    echo $_COOKIE['email'];
}
else {
?>
<input type="text" size="25" name="email" value="<? echo $_GET['email']; ?>" />
<?
}
if ($error_code && !($_GET['email'] || $_COOKIE['email'])) {
    echo "<b>Please include your email address.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Type of Request:
</td>
```

```

<td align="left">
<select name="whoami">
<option value="" />Please choose...
<option value="newcustomer">?
if ($_GET['whoami'] == "newcustomer") {
    echo " selected";
}
?> />I am interested in becoming a customer.
<option value="customer">?
if ($_GET['whoami'] == "customer") {
    echo " selected";
}
?> />I am a customer with a general question.
<option value="support">?
if ($_GET['whoami'] == "support") {
    echo " selected";
}
?> />I need technical help using the website.
<option value="billing">?
if ($_GET['whoami'] == "billing") {
    echo " selected";
}
?> />I have a billing question.
</select>
<?
if ($error_code && !($_GET['whoami'])) {
    echo "<b>Please choose a request type.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Subject:
</td>
<td align="left">
<input type="text" size="50" max="50" name="subject" value="<? echo $_GET['subject']; ?
>" />
<?
if ($error_code && !($_GET['subject'])) {
    echo "<b>Please add a subject for your request.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right" valign="top">
Message:
</td>
<td align="left">
<textarea name="message" cols="50" rows="8">
<? echo $_GET['message']; ?>
</textarea>
<?
if ($error_code && !($_GET['message'])) {
    echo "<b>Please fill in a message for us.</b>";
}
?>
</td>
</tr>
<tr>
<td colspan="2" align="left">
How did you hear about us?
<ul>
<input type="radio" name="found" value="wordofmouth" />Word of Mouth<br/>
<input type="radio" name="found" value="search" />Online Search<br/>
<input type="radio" name="found" value="article" />Printed publication/article<br/>
<input type="radio" name="found" value="website" />Online link/article<br/>

```

```

<input type="radio" name="found" value="other" />Other
</ul>
</td>
</tr>
<tr>
<td colspan="2">
<input type="checkbox" name="update1" checked="checked" />Please email me updates about
your products.<br/>
<input type="checkbox" name="update2" />Please email me updates about products from thi
rd-party partners.
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="SUBMIT" />
</td></tr>
</table>
</form>

<?
    require($_SERVER['DOCUMENT_ROOT']. "/template_bottom.inc");
?>

```

Be sure and **Save contact_form.php**, then Preview.

You should see something like this:

ACME, INC.

links go here

Contact ACME Corporation

Name: ☐ Remember me on this computer

Email:

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third-party partners.

Now, switch back to contact.php and make the following changes, in green:

```
<?php

function mail_message($data_array, $template_file, $deadline_str, $myname, $myemail) {

    #get template contents, and replace variables with data
    $email_message = file_get_contents($template_file);
    $email_message = str_replace("#DEADLINE#", $deadline_str, $email_message);
    $email_message = str_replace("#WHOAMI#", $data_array['whoami'], $email_message);
    $email_message = str_replace("#DATE#", date("F d, Y h:i a"), $email_message);
    $email_message = str_replace("#NAME#", $myname, $email_message);
    $email_message = str_replace("#EMAIL#", $myemail, $email_message);
    $email_message = str_replace("#IP#", $_SERVER['HTTP_X_FORWARDED_FOR'], $email_message);
    $email_message = str_replace("#AGENT#", $_SERVER['HTTP_USER_AGENT'], $email_message);

    $email_message = str_replace("#SUBJECT#", $data_array['subject'], $email_message);
    $email_message = str_replace("#MESSAGE#", $data_array['message'], $email_message);
    $email_message = str_replace("#FOUND#", $data_array['found'], $email_message);

    #include whether or not to contact the customer with offers in the future
    $contact = "";
    if (isset($data_array['update1'])) {
        $contact = $contact." Please email updates about your products.<br/>";
    }
    if (isset($data_array['update2'])) {
        $contact = $contact." Please email updates about products from third-party partners.<br/>";
    }
    $email_message = str_replace("#CONTACT#", $contact, $email_message);

    #construct the email headers
    $to = " ReplaceWithYourOwnEmailAddress@oreillyschool.com"; //for testing purposes,
    this should be YOUR email address.
    $from = $data_array['email'];
    $email_subject = "CONTACT #".time().": ".$data_array['subject'];

    $headers = "From: " . $from . "\r\n";
    $headers .= "MIME-Version: 1.0" . "\n";
    $headers .= "Content-type: text/html; charset=iso-8859-1" . "\r\n";    #now mail
    mail($to, $email_subject, $email_message, $headers);

}

$customer_name = $_COOKIE['name'];
if (!$customer_name) {
    $customer_name = $_GET['name'];
}
$customer_email = $_COOKIE['email'];
if (!$customer_email) {
    $customer_email = $_GET['email'];
}

#Remember, if you place any output before a header() call, you'll get an error.
#We used the superglobal $_GET here
if (!$customer_name && $customer_email && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";

    $query_string = $_SERVER['QUERY_STRING'];
    #add a flag called "error" to tell contact_form.php that something needs fixed
    $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1"
```

```

;
header("Location: ".$url);
exit(); //stop the rest of the program from happening
}

#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;

$deadline_stamp = mktime($deadline_array['hours'],$deadline_array['minutes'],$deadli
ne_array['seconds'],
    $deadline_array['mon'],$deadline_day,$deadline_array['year']);
$deadline_str = date("F d, Y", $deadline_stamp);

if (isset($_GET['remember'])) {
    #the customer wants us to remember him/her for next time
    ### set errcode cookie
    /*
        cookie expires in one year
        365 days in a year
        24 hours in a day
        60 minutes in an hour
        60 seconds in a minute
    */
    $mytime = time() + (365 * 24 * 60 * 60);
    setcookie("name",$customer_name,$mytime);
    setcookie("email",$customer_email,$mytime);
}

//DOCUMENT_ROOT is the file path leading up to the template name.
mail_message($_GET, $_SERVER['DOCUMENT_ROOT']."/email_template.txt", $deadline_str, $cu
stomer_name, $customer_email);

include($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

extract($_GET, EXTR_PREFIX_SAME, "get");

echo "<h3>Thank you!</h3>";
echo "We'll get back to you by ".$deadline_str."<br/>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #".time().":<br/>";
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
echo "Name: ".$customer_name."<br/>";
echo "Email: ".$customer_email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name.": ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWA
RDED_FOR'];

?>
<br/><br/><a href="download.php"><b>Download our PDF brochure!</b></a>
<?

```

```
include($_SERVER['DOCUMENT_ROOT']. "/template_bottom.inc");
```

```
?>
```

Save contact.php, switch to **contact_form.php**, and Preview. This time, however, when you submit the form, be sure to check the box that says "Remember me on this computer."

ACME, INC.

links go here

Thank you!

We'll get back to you by June 11, 2006.
Here is a copy of your request:

CONTACT #1149880113:

Message Date: June 09, 2006 02:08 pm

Name: Trish

Email: trish@myemail.com

Type of Request: support

Subject: Please help!

Message: I can't get the darn thing to work!

How you heard about us: wordofmouth

update1: on

update2:

You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X Ma
en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7

The IP address of the computer you're working on is 63.171.219.74

[Download our PDF brochure!](#)



Looks pretty much the same as before. What's changed? To find out, now go back to **contact_form.php** and **RELOAD** the page:

ACME, INC.

links go here

Contact ACME Corporation

Name: Trish

Email: trish@myemail.com

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third-party partners.

SUBMIT

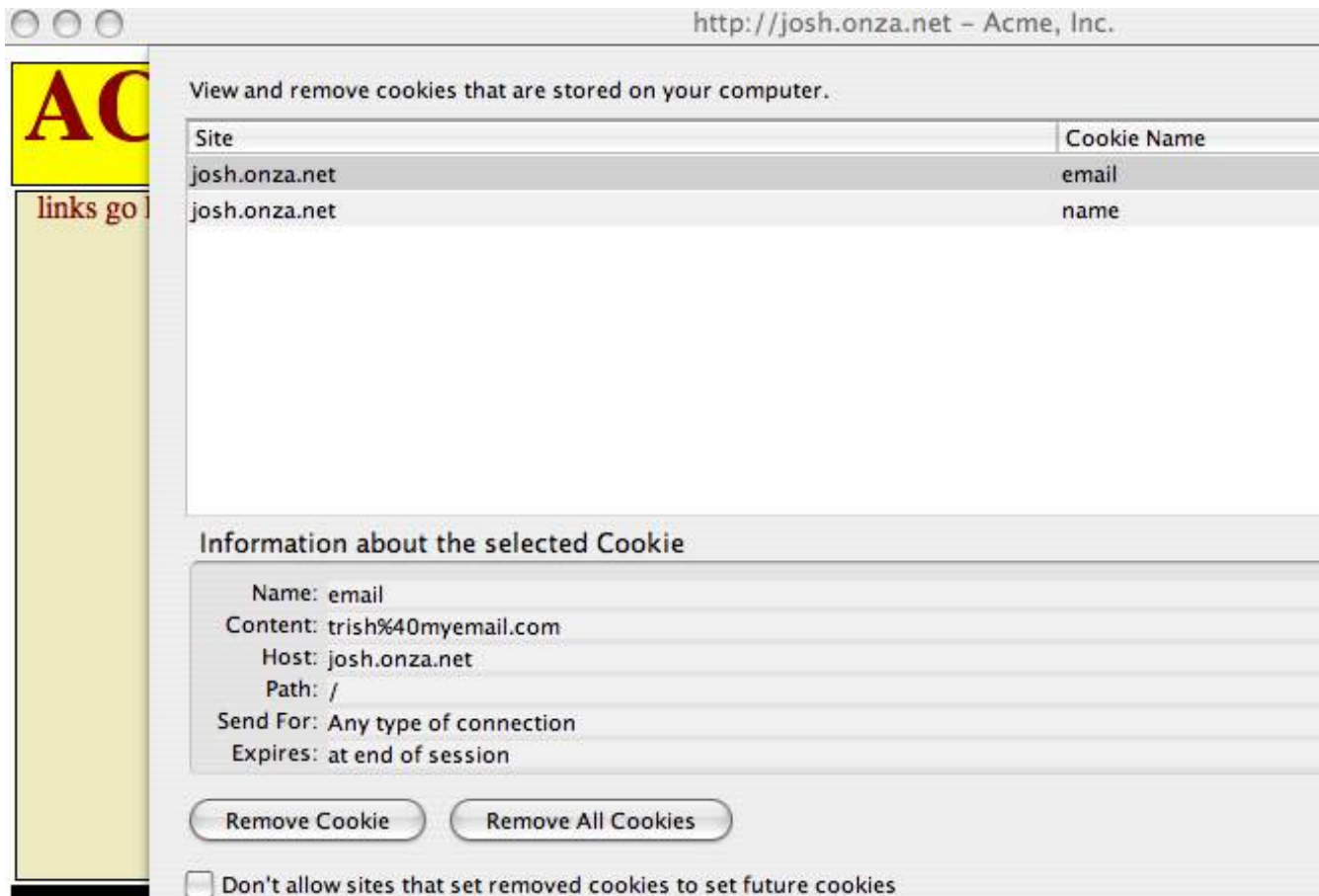
And there we are! The form is indeed remembering us, and even if you exit your browser entirely and come back, your name and email would still be there. But how were we able to do it? Using **cookies**.

Take another look at the code in contact.php:

```
if (isset($_GET['remember'])) {
    #the customer wants us to remember him/her for next time
    ### set errcode cookie
    /*
        cookie expires in one year
        365 days in a year
        24 hours in a day
        60 minutes in an hour
        60 seconds in a minute
    */
    $mytime = time() + (365 * 24 * 60 * 60);
    setcookie("name", $customer_name, $mytime);
    setcookie("email", $customer_email, $mytime);
}
```

Here, we're using the built-in PHP function `setcookie()` with three parameters: `"name"` and `"email"` are the names we're giving the respective cookies, and `$customer_name` and `$customer_email` are the values that we got from the `$_GET` superglobal. `$mytime` is the **timestamp** at which we want the cookies to expire - since it's measured in seconds, we simply took `time()` and added enough seconds to make 1 year.

Browser cookies are simply variables that are stored within the user's browser on his/her computer. If you look in your own browser preferences, you can actually view all the cookies that are set:



Now take another look at the code in `contact_form.php`

```
if ($_COOKIE['name']) {  
    echo $_COOKIE['name'];  
}
```

Just like `$_GET` and `$_POST` store values set by the user, and `$_SERVER` and `$_ENV` store values set by the environment, `$_COOKIE` is a **superglobal array** -- but this time the values being stored are set by you, the programmer.

Before cookies, once a user left a website, that site had no way recognizing that user when she came back. Basically, the user had to start from scratch every time. No shopping carts, personalized home pages, or pre-filled forms. So as you can see, introducing cookies opened up a world of power and convenience that have made them invaluable to web programming.

Knowing the User Through Sessions

Of course, there are a couple of downfalls to using cookies. One is that different browsers have different restrictions on the number and size of cookies - some allow unlimited numbers but small sizes, others allow large cookies but only up to 10.

But the main problem with cookies is privacy. Anyone who uses the same browser that you used - unless you deleted your cookies before you left - can now view your name and email in the browser cookie list. Think if that had been even more sensitive information, like usernames or financial information. Yikes! Let's try fixing this.

Add the following to contact_form.php, in green:

```
<?php

#start the session before any output
session_start();

require($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

if ($_GET['error'] == "1") {
    $error_code = 1; //this means that there's been an error and we need to notify the
customer
}

?>

<h3>Contact ACME Corporation</h3>
<?
if ($error_code) {
echo "<div style='color:red'>Please help us with the following:</div>";
}
?>
<form method="GET" action="contact.php">
<table>
<tr>
<td align="right">
Name:
</td>
<td align="left">
<?
if ($_SESSION['name']) {
    echo $_SESSION['name'];
}
else {
?>
<input type="text" size="25" name="name" value="<? echo $_GET['name']; ?>" />
<input type="checkbox" name="remember" /> Remember me on this computer
<?
}
if ($error_code && !($_GET['name'] || $_SESSION['name'])) {
    echo "<b>Please include your name.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Email:
</td><td align="left">
<?
if ($_SESSION['email']) {
    echo $_SESSION['email'];
}
else {
?>
<input type="text" size="25" name="email" value="<? echo $_GET['email']; ?>" />
<?
}
if ($error_code && !($_GET['email'] || $_SESSION['email'])) {
    echo "<b>Please include your email address.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
```

```

Type of Request:
</td>
<td align="left">
<select name="whoami">
<option value="" />Please choose...
<option value="newcustomer"<?
if ($_GET['whoami'] == "newcustomer") {
    echo " selected";
}
?> />I am interested in becoming a customer.
<option value="customer"<?
if ($_GET['whoami'] == "customer") {
    echo " selected";
}
?> />I am a customer with a general question.
<option value="support"<?
if ($_GET['whoami'] == "support") {
    echo " selected";
}
?> />I need technical help using the website.
<option value="billing"<?
if ($_GET['whoami'] == "billing") {
    echo " selected";
}
?> />I have a billing question.
</select>
<?
if ($error_code && !($_GET['whoami'])) {
    echo "<b>Please choose a request type.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Subject:
</td>
<td align="left">
<input type="text" size="50" max="50" name="subject" value="<? echo $_GET['subject']; ?
>" />
<?
if ($error_code && !($_GET['subject'])) {
    echo "<b>Please add a subject for your request.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right" valign="top">
Message:
</td>
<td align="left">
<textarea name="message" cols="50" rows="8">
<? echo $_GET['message']; ?>
</textarea>
<?
if ($error_code && !($_GET['message'])) {
    echo "<b>Please fill in a message for us.</b>";
}
?>
</td>
</tr>
<tr>
<td colspan="2" align="left">
How did you hear about us?
<ul>
<input type="radio" name="found" value="wordofmouth" />Word of Mouth<br/>
<input type="radio" name="found" value="search" />Online Search<br/>

```

```

<input type="radio" name="found" value="article" />Printed publication/article<br/>
<input type="radio" name="found" value="website" />Online link/article<br/>
<input type="radio" name="found" value="other" />Other
</ul>
</td>
</tr>
<tr>
<td colspan="2">
<input type="checkbox" name="update1" checked="checked" />Please email me updates about
  your products.<br/>
<input type="checkbox" name="update2" />Please email me updates about products from thi
rd-party partners.
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="SUBMIT" />
</td></tr>
</table>
</form>

<?
  require($_SERVER['DOCUMENT_ROOT']. "/template_bottom.inc");
?>

```

Be sure to **Save contact_form.php**.

Now switch to contact.php and make the following changes, in green:

```
<?php

function mail_message($data_array, $template_file, $deadline_str, $myname, $myemail) {

    #get template contents, and replace variables with data
    $email_message = file_get_contents($template_file);
    $email_message = str_replace("#DEADLINE#", $deadline_str, $email_message);
    $email_message = str_replace("#WHOAMI#", $data_array['whoami'], $email_message);
    $email_message = str_replace("#DATE#", date("F d, Y h:i a"), $email_message);
    $email_message = str_replace("#NAME#", $myname, $email_message);
    $email_message = str_replace("#EMAIL#", $myemail, $email_message);
    $email_message = str_replace("#IP#", $_SERVER['HTTP_X_FORWARDED_FOR'], $email_message);
    $email_message = str_replace("#AGENT#", $_SERVER['HTTP_USER_AGENT'], $email_message);

    $email_message = str_replace("#SUBJECT#", $data_array['subject'], $email_message);
    $email_message = str_replace("#MESSAGE#", $data_array['message'], $email_message);
    $email_message = str_replace("#FOUND#", $data_array['found'], $email_message);

    #include whether or not to contact the customer with offers in the future
    $contact = "";
    if (isset($data_array['update1'])) {
        $contact = $contact." Please email updates about your products.<br/>";
    }
    if (isset($data_array['update2'])) {
        $contact = $contact." Please email updates about products from third-party partners.<br/>";
    }
    $email_message = str_replace("#CONTACT#", $contact, $email_message);

    #construct the email headers
    $to = " ReplaceWithYourOwnEmailAddress@oreillyschool.com"; //for testing purposes,
    this should be YOUR email address.
    $from = $data_array['email'];
    $email_subject = "CONTACT #".time().": ".$data_array['subject'];

    $headers = "From: " . $from . "\r\n";
    $headers .= "MIME-Version: 1.0" . "\n";
    $headers .= "Content-type: text/html; charset=iso-8859-1" . "\r\n";    #now mail
    mail($to, $email_subject, $email_message, $headers);

}

#start the session
session_start();

$customer_name = $_SESSION['name'];
if (!$customer_name) {
    $customer_name = $_GET['name'];
}

$customer_email = $_SESSION['email'];
if (!$customer_email) {
    $customer_email = $_GET['email'];
}

#Remember, if you place any output before a header() call, you'll get an error.
#We used the superglobal $_GET here
if (!$customer_name && $customer_email && $_GET['whoami']
    && $_GET['subject'] && $_GET['message'])) {

    #with the header() function, no output can come before it.
    #echo "Please make sure you've filled in all required information.";
}
```

```

$query_string = $_SERVER['QUERY_STRING'];
#add a flag called "error" to tell contact_form.php that something needs fixed
$url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php?". $query_string."&error=1"
;

header("Location: ".$url);
exit(); //stop the rest of the program from happening

}

#we want a deadline 2 days after the message date.
$deadline_array = getdate();
$deadline_day = $deadline_array['mday'] + 2;

$deadline_stamp = mktime($deadline_array['hours'],$deadline_array['minutes'],$deadli
ne_array['seconds'],
    $deadline_array['mon'],$deadline_day,$deadline_array['year']);
$deadline_str = date("F d, Y", $deadline_stamp);

if (isset($_GET['remember'])) {
    #the customer wants us to remember him/her for next time
    $_SESSION['name'] = $customer_name;
    $_SESSION['email'] = $customer_email;
}

//DOCUMENT_ROOT is the file path leading up to the template name.
mail_message($_GET, $_SERVER['DOCUMENT_ROOT']."/email_template.txt", $deadline_str, $cu
stomer_name, $customer_email);

include($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

extract($_GET, EXTR_PREFIX_SAME, "get");

echo "<h3>Thank you!</h3>";
echo "We'll get back to you by ".$deadline_str."<br/>";
echo "Here is a copy of your request:<br/><br/>";
echo "CONTACT #".time().":<br/>";
echo "Message Date: ".date("F d, Y h:i a")."<br/>";
echo "Name: ".$customer_name."<br/>";
echo "Email: ".$customer_email."<br/>";
echo "Type of Request: ".$whoami."<br/>";
echo "Subject: ".$subject."<br/>";
echo "Message: ".$message."<br/>";
echo "How you heard about us: ".$found."<br/>";

for ($i = 1; $i <= 2; $i++) {
    $element_name = "update".$i;
    echo $element_name." : ";
    echo $$element_name;
    echo "<br/>";
}

echo "You are currently working on ".$_SERVER['HTTP_USER_AGENT'];
echo "<br/>The IP address of the computer you're working on is ".$_SERVER['HTTP_X_FORWA
RDED_FOR'];

?>
<br/><br/><a href="download.php"><b>Download our PDF brochure!</b></a>
<?

include($_SERVER['DOCUMENT_ROOT']."/template_bottom.inc");

?>

```

Save contact.php, then switch to contact_form.php and Preview. You'll notice that you have to re-enter your name

and email address again, but not for long. Be sure to click the "Remember me on this computer" checkbox when you submit the form. What did you get?

It should look something like this:

ACME, INC.

links go here

Thank you!

We'll get back to you by June 11, 2006.
Here is a copy of your request:

CONTACT #1149880113:
Message Date: June 09, 2006 02:08 pm
Name: Trish
Email: trish@myemail.com
Type of Request: support
Subject: Please help!
Message: I can't get the darn thing to work!
How you heard about us: wordofmouth
update1: on
update2:
You are currently working on Mozilla/5.0 (Macintosh; U; PPC Mac OS X I
en-US; rv:1.7.12) Gecko/20050915 Firefox/1.0.7
The IP address of the computer you're working on is 63.171.219.74

[Download our PDF brochure!](#)

Again, it looks exactly the same as always. But, if you go back to **contact_form.php** and *RELOAD*, you'll get:

Something like this:

ACME, INC.

links go here

Contact ACME Corporation

Name: Trish

Email: trish@myemail.com

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

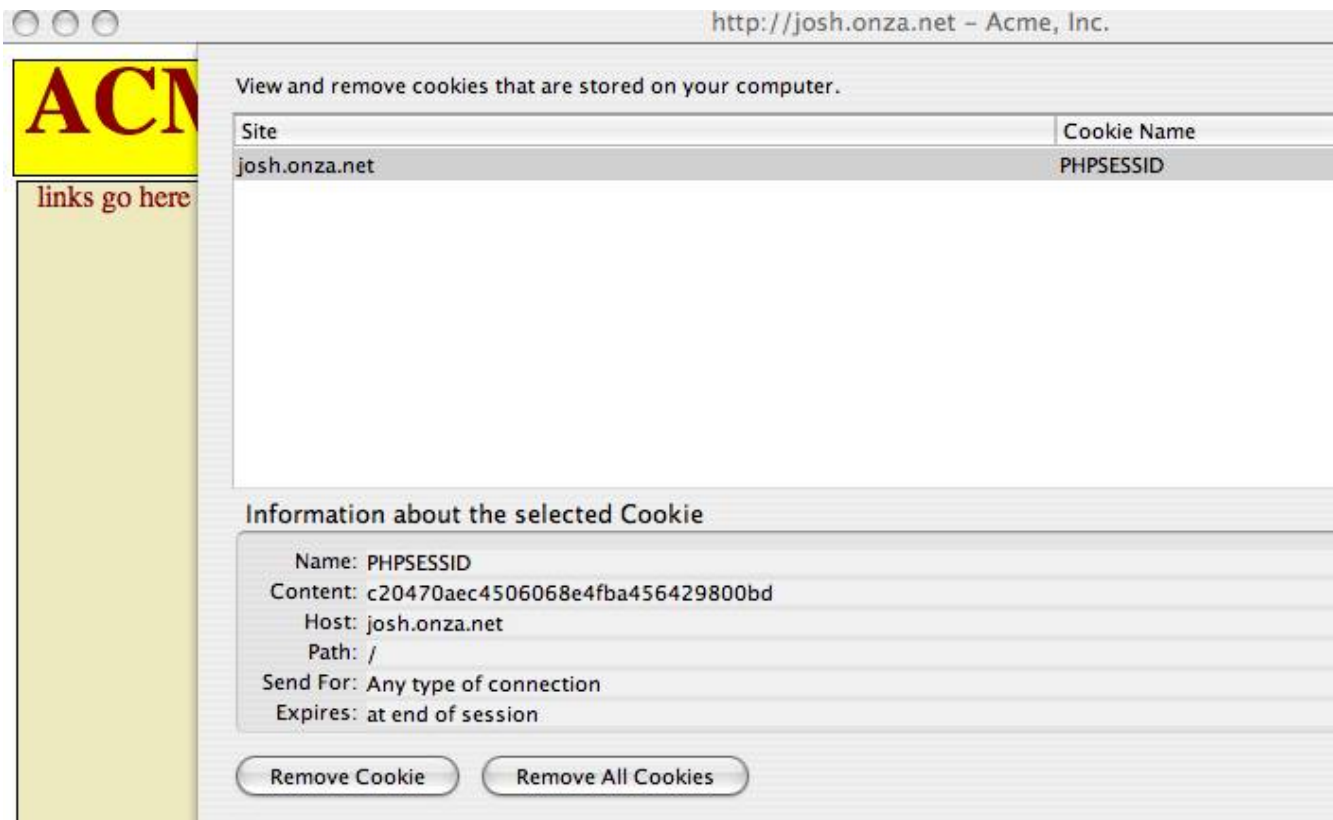
- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third party partners.

Yes, it's exactly the same output as when you used **cookies** -- your name and email address are now magically saved within the browser.

So what's the difference? If you check out your browser's preferences and view the cookies stored there, you won't see your name and email address in there anymore. Instead, you'll see something like this:



Take another look at the code in contact.php:

```
#start the session
session_start();

$customer_name = $_SESSION['name'];
if (!($customer_name)) {
    $customer_name = $_GET['name'];
}

$customer_email = $_SESSION['email'];
if (!($customer_email)) {
    $customer_email = $_GET['email'];
}
.
.
.
if (isset($_GET['remember'])) {
    #the customer wants us to remember him/her for next time
    $_SESSION['name'] = $customer_name;
    $_SESSION['email'] = $customer_email;
}
```

Any time you want to use **sessions** in your PHP script, you must start the session first - using the PHP function **session_start()**. This way, the browser knows to pull up the **\$_SESSION** **superglobal** using the **SESSION ID** that was set in your browser cookies. Once it's been pulled up, you can not only access the values using **\$_SESSION**, you can *set* the values too.

Note It's important to stress that **session_start()** must be called **before** any output - much like **header()**.

Deleting Sessions

In case someone else visits our site using the same browser, we should give the user a way to end the session without waiting for it to expire.

Add the following to contact_form.php, in green and blue:

```
<?php

if (isset($_GET['delete_session'])) {
    session_start(); //must always use this command to access the session and its
variables
    session_destroy(); //force the session to end

    //Add in a page reload so that the session_destroy() will take effect
    if($_SESSION && $_SESSION['name']){
        $url = "http://".$_SERVER['HTTP_HOST']."/contact_form.php";
        header("Location: ".$url);
    }
}
else {
    #start the session before any output
    session_start();
}

require($_SERVER['DOCUMENT_ROOT']."/template_top.inc");

if ($_GET['error'] == "1") {
    $error_code = 1; //this means that there's been an error and we need to noti
fy the customer
}

?>

<h3>Contact ACME Corporation</h3>
<?
if ($error_code) {
    echo "<div style='color:red'>Please help us with the following:</div>";
}
?>
<form method="GET" action="contact.php">
<table>
<tr>
<td align="right">
Name:
</td>
<td align="left">
<?
if ($_SESSION['name']) {
    echo $_SESSION['name'];
?>
    <a href="contact_form.php?delete_session=1">Not <? echo $_SESSION['name']; ?><
/a>
<?
}
else {
?>
<input type="text" size="25" name="name" value="<? echo $_GET['name']; ?>" />
<input type="checkbox" name="remember" /> Remember me on this computer
<?
}
if ($error_code && !($_GET['name'] || $_SESSION['name'])) {
    echo "<b>Please include your name.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Email:
</td><td align="left">
<?

```

```

if ($_SESSION['email']) {
    echo $_SESSION['email'];
}
else {
    ?>
<input type="text" size="25" name="email" value="<? echo $_GET['email']; ?>" />
<?
}
if ($error_code && !($_GET['email'] || $_SESSION['email'])) {
    echo "<b>Please include your email address.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Type of Request:
</td>
<td align="left">
<select name="whoami">
<option value="" />Please choose...
<option value="newcustomer"<?
if ($_GET['whoami'] == "newcustomer") {
    echo " selected";
}
?> />I am interested in becoming a customer.
<option value="customer"<?
if ($_GET['whoami'] == "customer") {
    echo " selected";
}
?> />I am a customer with a general question.
<option value="support"<?
if ($_GET['whoami'] == "support") {
    echo " selected";
}
?> />I need technical help using the website.
<option value="billing"<?
if ($_GET['whoami'] == "billing") {
    echo " selected";
}
?> />I have a billing question.
</select>
<?
if ($error_code && !($_GET['whoami'])) {
    echo "<b>Please choose a request type.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right">
Subject:
</td>
<td align="left">
<input type="text" size="50" max="50" name="subject" value="<? echo $_GET['subject']; ?>" />
<?
if ($error_code && !($_GET['subject'])) {
    echo "<b>Please add a subject for your request.</b>";
}
?>
</td>
</tr>
<tr>
<td align="right" valign="top">
Message:
</td>
<td align="left">

```

```

<textarea name="message" cols="50" rows="8">
<? echo $_GET['message']; ?>
</textarea>
<?
if ($error_code && !($_GET['message'])) {
    echo "<b>Please fill in a message for us.</b>";
}
?>
</td>
</tr>
<tr>
<td colspan="2" align="left">
How did you hear about us?
<ul>
<input type="radio" name="found" value="wordofmouth" />Word of Mouth<br/>
<input type="radio" name="found" value="search" />Online Search<br/>
<input type="radio" name="found" value="article" />Printed publication/article<br/>
<input type="radio" name="found" value="website" />Online link/article<br/>
<input type="radio" name="found" value="other" />Other
</ul>
</td>
</tr>
<tr>
<td colspan="2">
<input type="checkbox" name="update1" checked="checked" />Please email me updates about your products.<br/>
<input type="checkbox" name="update2" />Please email me updates about products from third-party partners.
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="SUBMIT" />
</td></tr>
</table>
</form>

<?
    require($_SERVER['DOCUMENT_ROOT']."/template_bottom.inc");
?>

```

Be sure to **Save contact_form.php**, then Preview.

It should look something like this:

ACME, INC.

links go here

Contact ACME Corporation

Name: Trish [Not Trish?](#)

Email: trish@myemail.com

Type of Request: Please choose...

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

☐ Please email me updates about products from third party vendors.

Try clicking the link to see what happens:

ACME, INC.

links go here

Contact ACME Corporation

Name: ☐ Remember me
computer

Email:

Type of
Request: Please choose...

Subject:

Message:

How did you hear about us?

- ☐ Word of Mouth
- ☐ Online Search
- ☐ Printed publication/article
- ☐ Online link/article
- ☐ Other

☒ Please email me updates about your products.

Ending the session was pretty straightforward, because `session_destroy()` will destroy all the session data for a user. If we wanted to delete just one session variable, we would use `unset($_SESSION['some_var'])`.

Congratulations! You've now learned the PHP skills needed to make a vast range of robust, commercial applications for the web. Are you ready for those skills to be tested? Make sure you have **Saved** your work and handed in the **assignments** for this lesson. Then, it's time for your **final project**.

Good luck! We know you can do it.

Copyright © 1998-2014 O'Reilly Media, Inc.



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.

Final Project

Final Project

The overall goal of this project is to create a shopping cart, with products, prices, registration, and a checkout area. You can make this shopping cart any way you wish.

For the sake of evaluation, try to include as many elements discussed in this course as you can. For instance, you should use arrays for products, functions for various program tasks, template files, form validation, and cookies/sessions for cart persistence. You are encouraged to observe good programming practices, with comments, code reusability and readability.

You can hand in up to five files, but you don't have to create that many if you don't want to.

Be creative and have fun! You want to present yourself in a professional yet friendly way, so feel free to express yourself!

Copyright © 1998-2014 O'Reilly Media, Inc.



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See <http://creativecommons.org/licenses/by-sa/3.0/legalcode> for more information.*