LAB 12a

INTRODUCTION TO PHP

What You Will Learn

- How to run PHP code using a web server
- How to make use of variables and conditionals in PHP
- How to write PHP functions

Note

This chapter's content has been split into two labs: Lab12a and Lab12b.

Approximate Time

The exercises in this lab should take approximately 50 minutes to complete.

Fundamentals of Web Development, 3rd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

XAMP version - Date Last Revised: February 7, 2022

SETTING UP PHP

PREPARING YOUR COMPUTER

The starting `lab12a` folder has been provided for you (within the zip folder downloaded from Gumroad).

Note: To complete the lab exercises that use PHP, you will have to have access to a web server. While the HTML files can be loaded from disk, the PHP files must be interpreted by the web server which then outputs HTML.

You may be using an online coding environment such as CodeAnywhere that already provides you with a PHP environment. You may have access to a server with PHP already. Alternatively, you may want to consider using an easy-to-set-up all-in-one tool like XAMPP, which is a lightweight version of Apache, PHP, and MySQL that can be installed in Windows or Mac OS X.

This lab cannot provide setup instructions for all the different possible student setup configurations and environments. Instead, this lab will provide instructions for using XAMPP

1. To set up XAMPP, download the XAMPP package for your operating system from their website. Run the installation package, accepting all defaults for the installation.

   Note, the instructions below assume you have installed XAMPP in the default location.

2. Windows: start the XAMPP control panel (if not available from your computer's Start menu, it is the file `xampp-control.exe` in your XAMPP folder). Click Start next to Apache. A green box will show it is running.

   Mac: double-click the XAMPP icon in Applications) and click the Start button in the General tab on Mac). Click on the Services tab and ensure Apache has a green circle beside it. Click on the Network tab and enable localhost:8080 (click on option then click Enable button). There should be a green circle beside localhost:8080.

   Figure 12.1 illustrates the XAMPP control panel on Windows and Mac

3. Create a folder named `lab12a` in your `htdocs` folder.

   Windows: clicking the Explorer button in XAMPP will open the root folder for XAMPP (in Windows `c:/xampp/`) in File Explorer. Copy the contents of the provided lab folder into the `lab12a` folder.

   Mac: you first have to click on Volumes tab in XAMPP, then click Mount. You can then click the Explore button to see a Finder window showing the files and folders in XAMPP, including the `htdocs` folder. Copy the contents of the provided lab folder into the `lab12a` folder.

   If you are using XAMPP in a lab environment, you will need to remember to move your completed work back to your personal drive location!

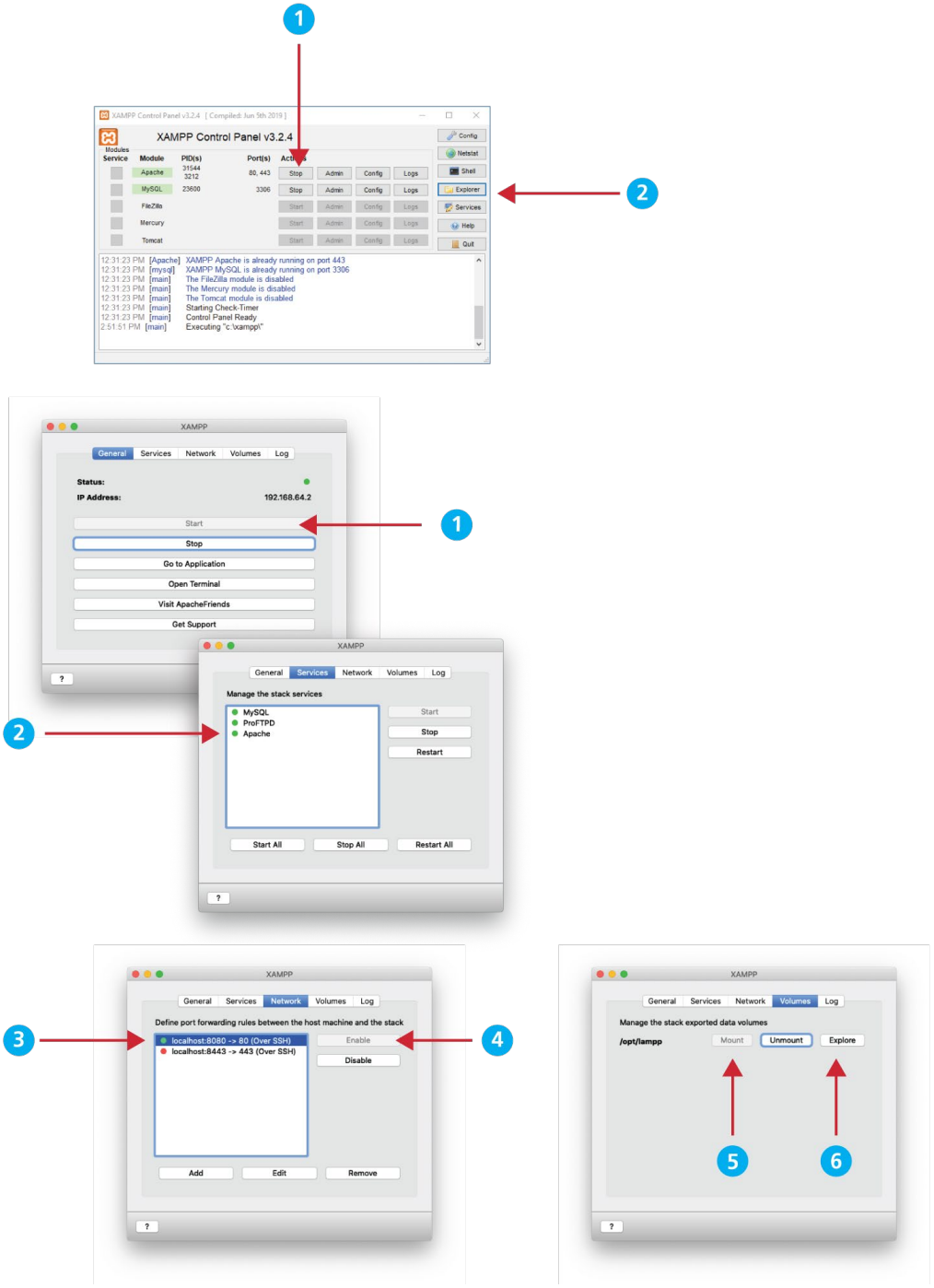

Figure 12.1 –Configuring XAMPP on Windows + Mac

USING PHP

EXERCISE 12.1 — TESTING YOUR CONFIGURATION

1. You have been provided with a file called `lab12a-ex01.php`. You can examine it in your editor. It simply contains one line of code that outputs information about your PHP environment.

2. Request the `lab12a-ex01.php` page on your server. How you do this depends on your environment. You can't just open the file in your browser since a PHP page needs to be requested from (i.e., executed on) a server.

   If you are using XAMPP and it is running, then you will need to request the file via one of the following:

   `http://localhost/lab12a/lab12a-ex01.php` (Windows)

   `http://192.168.64.2/lab12a/lab12a-ex01.php` (Mac)

   The result should look similar to Figure 12.2.

3. Within the browser, choose the View Source option.

   Notice that you see just HTML: you don't see any PHP.

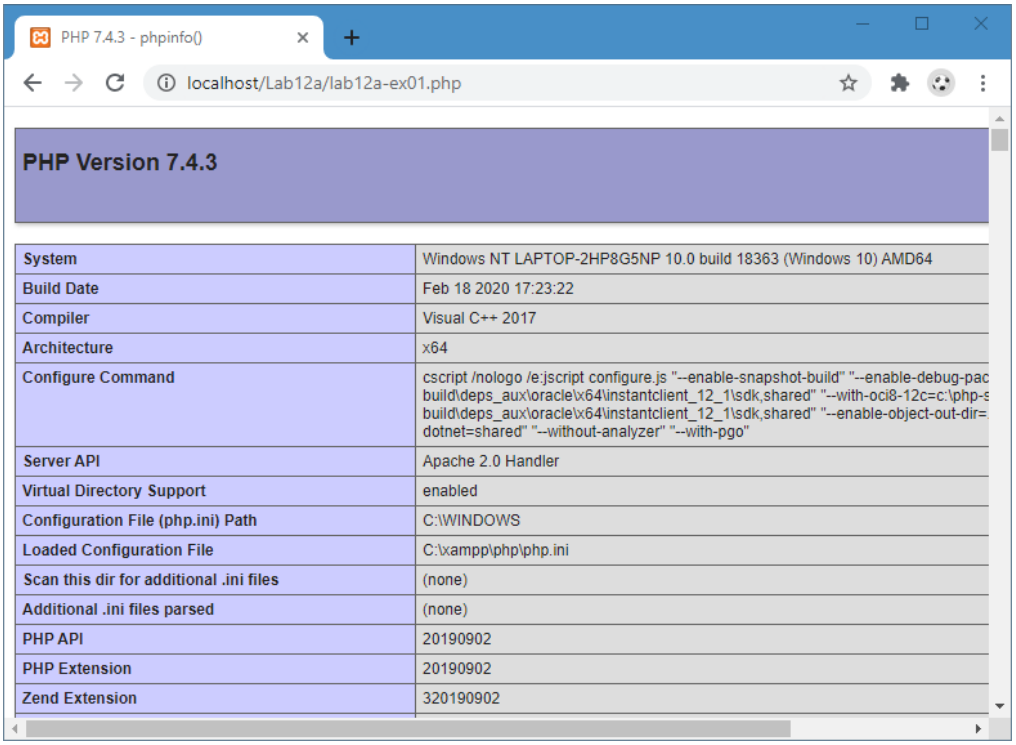

Figure 12.2 –phpinfo() is interpreted, rather than output as text.

EXERCISE 12.2 — YOUR FIRST PHP SCRIPT

1. Examine `lab12a-ex02.php` in a text editor and then load it in a browser (that is, using the same technique as in the previous exercise). You should see that the PHP `echo` statement prints out a string and that an `echo` statement outside of the PHP tags does not get executed.

2. Add an `echo` statement inside the PHP tags that outputs the data and time.

```
echo "This page was generated: " . date("M dS, Y");
```

The dot operator in PHP is used to concatenate values. Your page will now also output a String like: Jun 19th, 2019

3. Examine the markup received within the browser (e.g., View Source).

PHP is processed by the server and is not sent to the browser.

4. Modify the code you just entered as follows and then test in browser.

```
echo "This page was generated: " . date("M dS, Y") . "<hr/>";
```

Notice that we can programmatically output HTML via the `echo` command.

5. Modify the previous line by removing one of the dot operators and test.

This will generate an error. As long as your error reporting settings are at the default setting, you should see a syntax error message in the browser along with an indication of the line number of the error.

6. Fix the syntax error and retest.

7. Modify the code as follows and then test in browser.

```
$d = date("M dS, Y");
echo "This page was generated: " . $d . "<hr/>";
```

You should notice no change in the browser. This new version differs in that it uses a variable (`$d`). Variables can be named anything but must begin with the $ symbol.

8. Modify the code as follows and then test in browser.

```
$date = date("M dS, Y");
echo "This page was generated: " . $date . "<hr/>";
```

You should again notice no change in the browser. The new variable name (`$date`) is to remind you that variables begin with the $ symbol, while functions have brackets after their name. So, in this example, the variable `$date` is assigned the value returned by the function `date()`.

9. There are a variety of ways that one can use PHP to generate markup. Add the following code and test.

```
$filename = "134020.jpg";
echo "<img src='images/" . $filename . "'>";
```

10 Comment out the previous line and replace it with the following and test (it should work exactly the same).

```
echo "<img src='images/$filename'>";
```

PHP allows you to embed variable names within string literals that use double quotes.

11 Comment out the previous line and replace it with the following and test:

```
$img = "<img src='images/$filename'>";
echo $img;
```

12 Make the following change and test. It should generate an error message.

```
echo $tag;
```

13 Comment out the previous line, add the following markup, and test:

```
$img = "<img src='images/$filename'>";
//echo $img;
?>
<img src='images/<?php echo $filename; ?>'>
```

Instead of echoing an entire HTML element, it is often preferable to "inject" PHP data into your markup.

14 Add the following markup.

```
<img src='images/<?=$filename ?>'>
```

This uses the alternate shorthand syntax (known as short tags) for displaying PHP variables. This is often the preferred way to output PHP variables.

15 Modify the markup as follows (space added between ? and =) and test. It won't work.

```
<img src='images/<? =$filename ?>'>
```

16 Modify the markup as follows and test. It works.

```
<img src='images/<?= $filename ?>'>
```

Clearly it is a little fiddly!

Test Your Knowledge #1

Open `lab12a-test01.php` in your editor.

Notice that this file already has defined within it several PHP variables. As you progress through the book, such variables will later be populated from arrays, files, and then databases.

1. Use the PHP `echo` statement to output the relevant PHP variables (or the `<?= ?>` shorthand) so that your page looks similar to that shown in Figure 12.3. Note: the CSS styling has already been provided
2. Try experimenting with different ways of using PHP to outputting the image

Figure 12.3–Completed lab12a-test01.php

Exercise 12.3 — PHP Operators

1. Open the file `lab12a-ex03.php` in an editor of your choice.
2. Inside you will find some initial HTML to output a message regarding a person's age.

   Create a new set of `<?php ?>` tags after the `<h1>` element. Define two variables inside of the PHP tags as follows.

```
<?php
$birthday = mktime(0,0,0,1,15,2004); //Jan 15, 2014 00:00:00
$today = time(); // current time in seconds since 1970.
?>
```

   You can enter any date that interests you, for instance, your birthday. The parameters to `mktime()` are: Hours, Minutes, Seconds, Month, Days, Year

3. You can calculate the time elapsed from the first date (the birthday) to the present day in seconds by subtracting them and storing the result in an new variable:

```
$secondsOld = $today - $birthday;
```

4 Echo the starting date before the `<ul>` in its own `<p>`:

```
echo "<p>Time elapsed since " . date("M d, Y",$birthday) .
":</p>";
```

5 Now calculate and display how many days, months and years those seconds represent. For instance, to calculate the age in days you would output:

```
echo $secondsOld/(60*60*24);
```

6 Calculate the number of elapsed months and years so you get output similar to Figure 12.3. For simplicity sake, assume 30.4 days per month and 365.242375 days per year.

Your numbers will be different than that shown in Figure 12.4. Why?

Figure 12.4 - Completed EXERCISE 12.3 – Your numbers will be different!

EXERCISE 12.4 — PHP OUTPUT

1 The last exercise demonstrated some simple calculations, but the output of the numbers suffered, since many decimal places were showing. Continue working on the file from Exercise 12.3.

2 Replace the number of days, months, and years using the `number_format()` function. Consult `http://php.net/number_format` for more information.

Number of second and days should have no decimal places; months should have one and years two, as shown in Figure 12.5.

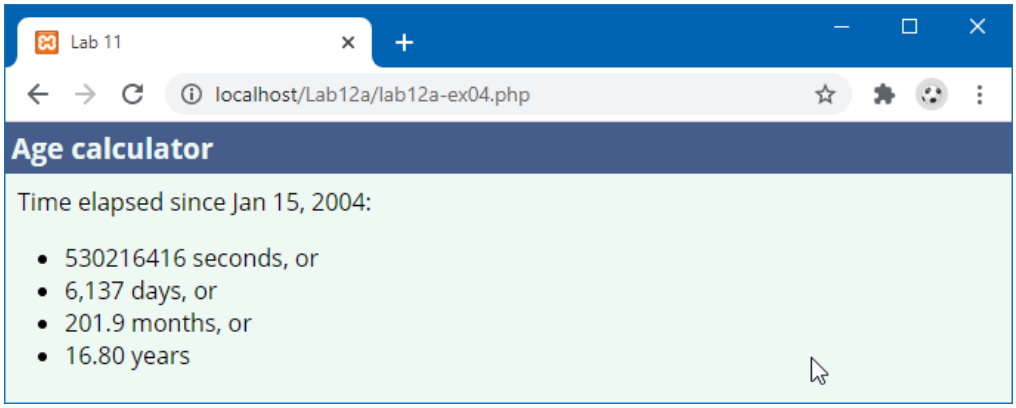

Figure 12.5 – Completed Exercise 12.5 – Your numbers will be different!

Program Control

Exercise 12.5 — PHP Conditionals

1 In `lab12a-ex05.php`, add the following alternate approach to concatenating string literals with variable data:

```
$thumbnail = "134020.jpg";
$title = "Portrait of Isabella of Portugal ";
$img =  "<img src='images/$thumbnail' alt='$title'
              title='$title' />";
```

If a string literal uses double quotes, you can specify PHP variables within it; PHP will then replace the variable identifier with the contents of that variable.

2 Add the following and test.

```
<?= $img ?>
<h2><?=$era?></h2>
```

This uses the alternate shorthand syntax for displaying PHP variables. Notice that spaces are optional!

3 Modify the variable block in your code and add the following conditional statement.

```
$era = "15th Century";
if ($year > 1500)
   $era = "16th Century ";
```

4 Test in browser. Why didn't the output change?

It didn't change because the hard-coded year is not less than 1500 (it's equal).

5 Change the value of the `$year` variable to 1510 and test. The era should now display as 15th Century.

6 Change the value of the `$year` variable back to 1500 and change the comparison in the condition to `>=` and test.

7 Add the following, change the `$year` to `1450`, and test.

```
$year = 1450;
$era = "15th Century";
if ($year >= 1500)
   $era = "16th Century";
else
   $era = "17th Century";
```

Can you figure out why it displays 17th century and not 15th century?

8 Change the `$year` to `1800` and test.

Why does it display 16th century? The reason is due to how program flow works within conditionals. The number 1800 is indeed greater than the number 1500, so the condition `$year > 1500` evaluates to true, and thus the era is set to 16th century.

9 Change the code as follows and test using a variety of year dates.

```
$year = 1850;
if ($year < 1500) {
    $era = "Early times";
} else if ($year >= 1500 && $year < 1600) {
   $era = "16th Century";
} else if ($year >= 1600 && $year < 1700) {
   $era = "17th Century";
} else if ($year >= 1700 && $year < 1800) {
   $era = "18th Century";
} else {
   $era = "Modern times";
}
```

EXERCISE 12.6 — PHP LOOPS

1 Loops in PHP lend themselves nicely to building lists and tables. Examine `lab12a-ex06.php`, and when you test it, you will see that it currently outputs a series of pagination links with hard-coded start and ending numbers.

2 Replace the hard-coded list of `<a>` elements with the following PHP loop and test.

```
<div class="pagination">
<?php
for ($i=1; $i<7; $i++) {
    echo "<a href='#'>$i</a>";
}
?>
</div>
```

The result in the browser should be the same (except we've lost the special formatting for the active page (formerly the second item).

3 Let's make the code more generalized by making the starting and ending numbers into variables by adding the following code and test.

```
<?php
$start = 10;
$end = 21;
for ($i=$start; $i<$end; $i++) {
    echo "<a href='#'>$i</a>";
}
?>
```

4 Now let's add in the ability to add the `active` CSS class to an item by adding the following and test. The result should look similar to that shown in Figure 12.6.

```
<?php
$start = 10;
$end = 21;
$active = 16;
for ($i=$start; $i<$end; $i++) {
    echo "<a href='#' ";
    if ($i == $active) echo "class='active'";
    echo ">$i</a>";
}
?>
```

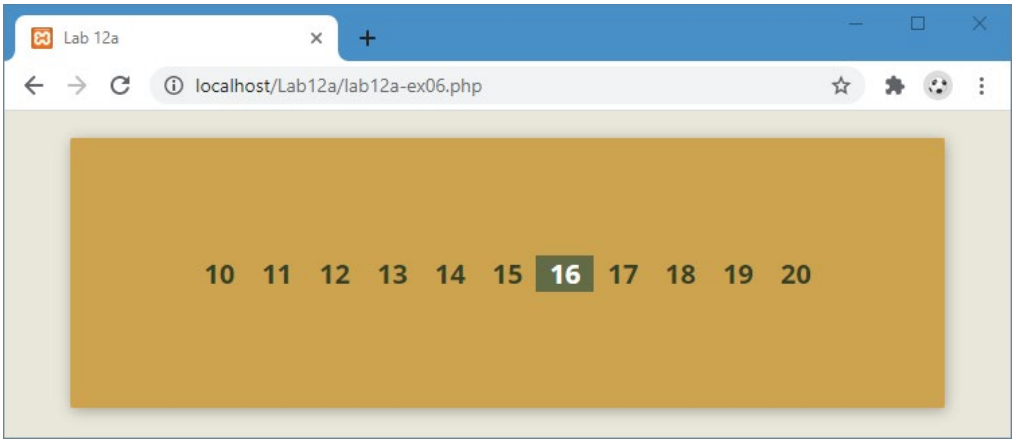

Figure 12.6 – Exercise 12.6 complete.

EXERCISE 12.7 — WRITING FUNCTIONS

1 Functions allow us to group code into modules that can be reused. In this example you will begin by writing functions that to convert between imperial and metric units for temperature and speed.

Begin by examining `lab12a-ex07.php` in the editor and then in the browser.

You are going to replace the hard-coded values with calculated values in PHP.

2 Create a new file named `lab12a-ex07.inc.php` and add the following code.

```
<?php
// returns the Celsius equivalent of the passed Fahrenheit value
function convertF2C($fahr) {
    return ( ($fahr - 32) * 5 ) / 9;
}
// returns the kpm equivalent of the passed mph value
function convertK2M($mph) {
    return $mph * 1.609344;
}
?>
```

3 At the top of `lab12a-ex07.php` add the following PHP code block:

```
<?php
include 'lab12a-ex07.inc.php';
?>
<!DOCTYPE html>
<html>
<head>
```

4 Replace the first hard-coded Celsius and kpm values with function invocations as shown below. Test.

```
<div>October 8</div>
<div><i class="fas fa-sun"></i></div>
<div><span><?= convertF2C(73) ?>°C | 73.4°F</span></div>
<div><?= convertK2M(1.2) ?> kmh | 1.2 mph</div>
```

You will likely see a very long string of decimal digits. You can use the `number_format()` function from EXERCISE 12.4 to fix them. The question is, where should you use it? In the function definition of after we call it?

5 Edit `lab12a-ex07.inc.php` as follows and test.

```
// returns the Celsius equivalent of the passed Fahrenheit value
function convertF2C($fahr) {
    return number_format( (($fahr - 32) * 5 ) / 9, 1);
}
// returns the kpm equivalent of the passed mph value
function convertK2M($mph) {
    return number_format($mph * 1.609344, 1);
}
```

The number of decimal digits should now be just one.

6 Replace the other hard-coded Celsius and kpm values with function invocations and test.

7 Notice that there is quite a bit of markup and code duplication in our page. You can also define functions that do things (such as echo markup) in order to reduce markup duplication in your pages (and thus improve maintainability).

Define the following function in `lab12a-ex07.inc.php`:

```
// outputs a single weather row with passed data
function generateWeatherRow($date, $symbol, $fahr, $mph) {
    echo "<div>$date</div>";
    echo "<div><i class='fas fa-$symbol'></i></div>";
    echo "<div><span>" . convertF2C($fahr) . "° C | " . $fahr .
            "° F</span></div>";
    echo "<div>" . convertK2M($mph) . " kmh | $mph mph</div> ";
}
```

8 Replace the markup for the weather forecasts with calls to this new function and test:

```
<div class="weather">
    <h2>Calgary</h2>
    <h4>Temp</h4>
    <h4>Wind</h4>
    <?php

    generateWeatherRow("October 8", "sun", 73.4, 1.2);
    generateWeatherRow("October 9",  "snowflake", -9.4, 26.6);
    generateWeatherRow("October 10", "snowflake", 17.6, 4.4);
    generateWeatherRow("October 11", "cloud", 37.4, 15.3);
    generateWeatherRow("October 12", "sun", 55.4, 1.9);

    ?>
</div>
```

Everything should work correctly and look similar to that shown in Figure 12.7.

9 The one problem with our existing function is that it contains quite a lot of markup within the code, which can decrease maintainability. Try modifying the function to use this alternate approach (some existing code omitted):

```
<?php
function convertF2C($fahr) {
  ...
}
function convertK2M($mph) {
  ...
}
function generateWeatherRow($date, $symbol, $fahr, $mph) { ?>

   <div><?php echo $date; ?></div>
   <div><i class='fas fa-<?php echo $symbol; ?>'></i></div>
   <div><span><?php echo convertF2C($fahr); ?>° C |
      <?php echo $fahr; ?>° F</span></div>
   <div><?php echo convertK2M($mph); ?> kmh | <?php echo $mph;
?> mph</div>

<?php } // end function
?>
```

This might seem less understandable than the previous version of the function. But imagine if each weather row was dozens of lines of markup. In such a case, keeping the markup as markup (that is, not putting the markup within PHP string literals) is much preferred. In fact, we can use a shorter syntax to make it even better.

10 Modify the markup in the revised `generateWeatherRow()` function as follows and test.

```
<div><?=$date?></div>
<div><i class='fas fa-<?=$symbol?>'></i></div>
<div><span><?=convertF2C($fahr)?>° C | <?=$fahr?>°
F</span></div>
<div><?=convertK2M($mph)?> kmh | <?=$mph?> mph</div>
```

This syntax improves the readability of the injection of PHP values into our markup.

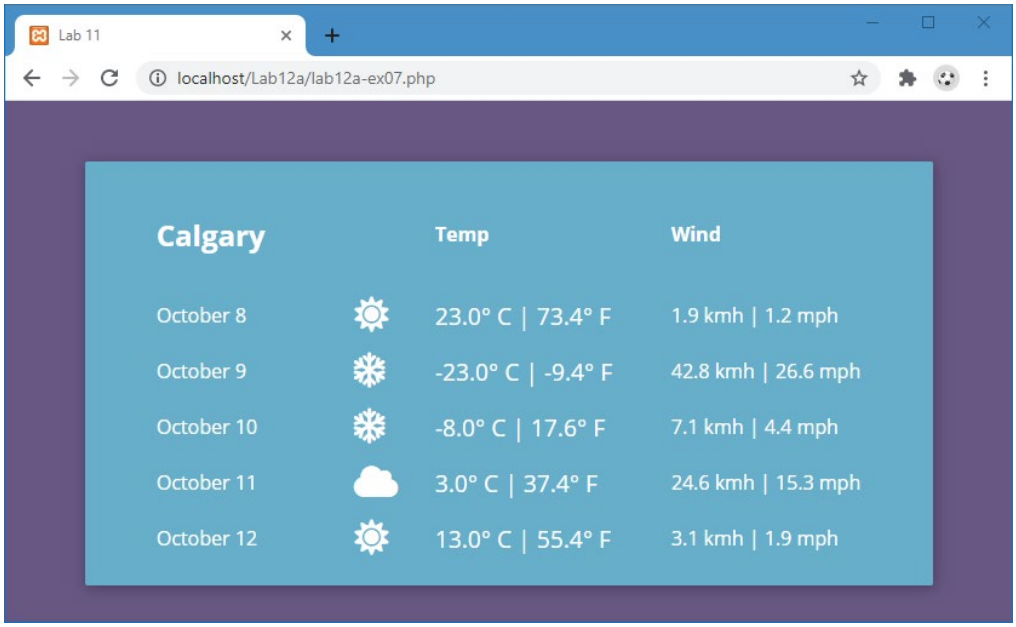

Figure 12.7 – Completed Exercise 12.7

EXERCISE 12.8 — SCOPE IN PHP

1 Variable scope in PHP is unlike that in JavaScript. This exercise illustrates that difference. Add the following to `lab12a-ex08.php` then test.

```
<?php
$outside = "I'm outside of any functions";
echo "Outside=" . $outside . "<br/>";

function testScope($param) {
   $inside = "I'm inside a function";
   echo "Inside=" . $inside . "<br/>";
   echo "Param=" . $param . "<br/>";
   echo "Outside=" . $outside . "<br/>";
}
// now invoke that function
testScope("I'm a parameter");
?>
```

This will generate a runtime error since $outside is undefined within a function.

2 Modify your code as follows (notice you need to comment out two lines) and test.

```
function testScope($param) {
   $inside = "I'm inside a function";
   echo "Inside=" . $inside . "<br/>";
   echo "Param=" . $param . "<br/>";
   //echo "Outside=" . $outside . "<br/>";
}
// now invoke that function
//testScope("I'm a parameter");
testScope($outside);
```

This illustrates the usual solution to function scope in PHP: pass in the value to the function as a parameter.

TEST YOUR KNOWLEDGE #2

Open `lab12a-test02.php` in your editor and examine in browser.

Notice that this file already has the markup. You will have to replace markup with appropriate PHP codes.

1. Create two functions: one that converts a US dollar amount to a Euro amount; the other converts from US dollar to UK pound. Each of these should return a numeric value with no decimals and rounded up. Feel free to use the current conversion rate: for the numbers in the screen capture, the rates were $1= €.87 and $1= £0.76.

   Be sure to first define PHP constants for these exchange rates (see `http://php.net/manual/en/language.constants.php`).

2. Create a function called `generateBox()` that generates the markup for a single pricing box. It must only have the following parameters: name and number of users.

   The other data values can be calculated within this function from those parameters.

   Notice that the calculation for cost, storage, and number of emails is different for the professional and enterprise boxes, which will require the use of conditional logic. There is a 10% discount for ten users, and a 20% discount for 50 users.

   Be sure to define these functions in an external file called `lab12a-test02.inc.php`.

3. Remove the markup for the boxes and replace them with invocations of your `generateBox()` function. Be sure to include your function file.

   The page should look similar to that shown in Figure 12.8.

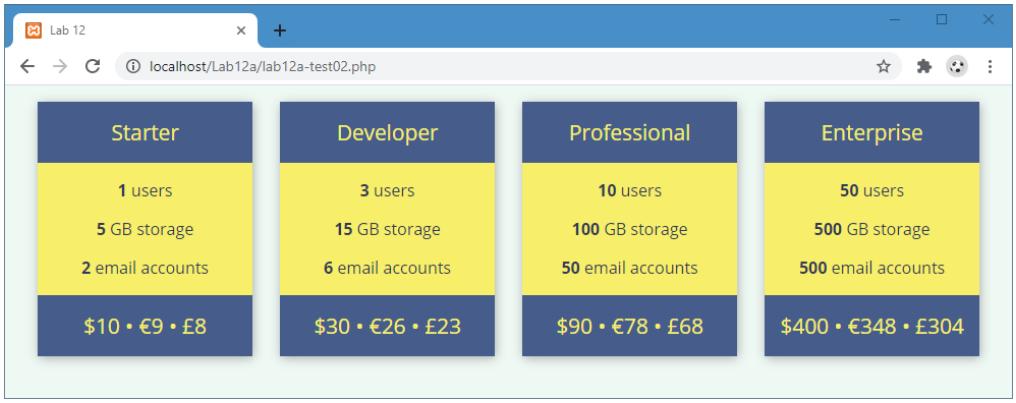

Figure 12.8 – Completed Test Your Knowledge #2

NOTE: If you are using XAMPP in a shared lab environment, remember to move your completed work from `c:/xampp/htdocs` back to your personal drive location!!