Hands-On Exercises #4 – PHP script

Source: Connolly, R. and Hoar, R., (2018), Fundamentals of Web Development

Before you do the exercises

If you haven't already done so, you should create a folder in your personal drive for all the exercises. This set of exercises requires a functioning webserver to interpret the PHP code. While the HTML files can be loaded from disk, PHP files must be interpreted by the web server which then outputs HTML

Exercise 4-1 First PHP Script

1. Examine `lab04-exercise01.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 of 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 13th, 2021

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 using 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. Experiment with the string passed into the date() function. See if you can make the following formatted string (note the day will be today)

   ```
   Sunday, June 13th , 2021 11:26:43
   ```

   You will need to make use of documentation from:
   https://www.php.net/manual/en/function.date.php

10. To calculate the number of days remaining in the year, consider the format string "z" passed to the date() function. For instance, date("z") returns the numbers of days elapsed this year, so we can calculate how many days are remaining by subtracting it from 365.

    ```
    $remaining = 365 - date("z");
    echo "There are ". $remaining . " days left in the year";
    ```

11. The above calculation does not work if the current year is a leap year. Let us pretend that the current year is a leap year by correcting the above calculation to account for leap years. *Hint: leap years have an extra day in the year. Simply add one to the calculation!*

Exercise 4-2 PHP Loops

1. Loops in PHP lend themselves nicely to building lists and tables. Open and examine `lab04-exercise02.php`, and see that it currently outputs a simple table with the days of the week for headers.

2. Using the `date()` function, determine the current month as a word (e.g., February). Weave that variable into the HTML above the table (maybe with an extra colspan row). Your first attempt at building a calendar will be by looping through all 31 days and outputting each in a <td> cell. Every 7 days, we will add a new row to the table. Notice the use of the mod or % operator.

   ```
   $day = 0;
   $dayCount=0;
   while ($day<=31) {
      //when we need a new row go ahead.
      if ($day%7==0) {
         echo "<tr></tr>";
      }
      echo "<td>".($day+1)."</td>";
      $day++;
      $dayCount++;
   }
   ```

   You should see output similar to the table shown in Figure 4.1 below.

November						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figure 4.1 Table produced after step2 was completed. Note the number of days being 31 for November is not correct. This will be fixed in steps 3 and 4.

3. Since most months don't start on Sunday, and they don't all have 31 days we will now improve our loop to account for those things. Firstly, determine how many days are in the month, and change our loop to loop only that many times. Hint: Change the constant 31, to a value determined by the date function.

```
$day = 0;
$dayCount=0;
$daysinMonth = date("t");
while ($day<=$daysinMonth) {
   //when we need a new row go ahead.
   if ($day%7==0) {
      echo "<tr></tr>";
   }
   echo "<td>".($day+1)."</td>";
   $day++;
   $dayCount++;
}
```

Now, the program should produce correct number of days for a month similar to Figure 4.2

November						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Figure 4.2 Number of days is not fixed.

4. Determine which day of the week, the 1st is as follows:

```
$day = 0;
$dayCount=0;
$dayOne = date("w",mktime(0,0,0,date("n"),1, date("Y")));
```

```
$daysinMonth = date("t");
while ($day<=$daysinMonth) {
   //when we need a new row go ahead.
   if ($day%7==0) {
      echo "<tr></tr>";
   }
   echo "<td>".($day+1)."</td>";
   $day++;
   $dayCount++;
}
```

The value, 0-6 tells us if the first was Sunday (0) through Saturday (6). Add a flag to our earlier loop so that the first time through we print empty cells (<td></td>) until we reach the first day of the month. Your output table should look like
that in Figure 4.3

```
$day = 0;
$started=false;
$dayCount=0;
$dayOne = date("w",mktime(0,0,0,date("n"),1, date("Y")));

$daysinMonth = date("t");
while ($day<=$daysinMonth) {
   //when we need a new row go ahead.
   if ($day%7==0) {
      echo "<tr></tr>";
   }
   if(!$started){
      if($dayCount==$dayOne){
        $started=true;
        echo "<td>".($day+1)."</td>";
      }
      else {
        echo "<td></td>";
        $dayCount++;
        continue;
      }
    }
    else {

      echo "<td>".($day+1)."</td>";
   }
   $day++;
   $dayCount++;
}
```

November						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Figure 4.3 Completed Exercise 4-2

Exercise 4-3 PHP Functions (Optional for this week)

1. Functions allow us to group code into modules that can be reused. In this case we will write a function that converts between various kitchen units (cups, teaspoons, etc). Begin by opening `lab04-exercise03.php`

2. Inside the empty `<?php ?>` tags define a function named `convertUnits()` that takes 3 parameters. To begin make the function look like:

```
function convertUnits($startVal, $startUnits, $endUnits) {
      return "???";
}
```

3. Now write code to handle converting values from millilitres to cups and ounces. Add the following constants to your new function.

```
$mlToOz = 0.033814;
$mlToCup = 0.00422675;
```

Also add conditional logic to ensure that the starting units are "ml", and the end values are either "cups" or "oz".

4. Integrate the function so that it gets called each time though the loop with the correct parameters. Since $i represents millilitres, you should be calling the function as follows:

```
for($i=50;$i<=1000;$i+=50){
   echo "<tr>";
   echo "<td>$i</td>";
   // replace the ??? with the calls to convertUnits function
   echo "<td>" . convertUnits($i,"ml","cups") . "</td>";
   echo "<td>" . convertUnits($i,"ml","oz") . "</td>";
   echo "</tr>";
}
```

5. Now run your program and your table of output should resemble that in Figure 4.2. Notice that the cups and ounces values have only two decimal values of precision.

Making and using functions

milliliters	Cups	Ounces
50	0.21	1.69
100	0.42	3.38
150	0.63	5.07
200	0.85	6.76
250	1.06	8.45
300	1.27	10.14
350	1.48	11.83
400	1.69	13.53
450	1.90	15.22
500	2.11	16.91
550	2.32	18.60
600	2.54	20.29
650	2.75	21.98
700	2.96	23.67
750	3.17	25.36
800	3.38	27.05
850	3.59	28.74
900	3.80	30.43
950	4.02	32.12
1000	4.23	33.81

Figure 4.2 Completed Exercise 4-3