PHP Homework for Week 5 : Strings

Reminder

Do not copy code from any MS Word file, PowerPoint file, or PDF file (such as this one) into your Brackets file. Always type the code in yourself.

Exercise 1: Comparing Strings

1. Create a folder in your web server directory. Name the folder `week05`.
2. In the week05 folder, create a partial HTML file with DOCTYPE, html, head, body, meta charset, and title tags. Save the file as `compare.php`. Your code might look like this.
3. Add a PHP script section to the document body:

```
<?php
?>
```

4. In the script section, initialize two string variables:

```
$string1 = "California";
$string2 = "Virginia";
```

5. Add the following PHP code to the script section. If two strings are declared, and they are not the same, this code will compare them using the levenshtein and similar_text methods, and print out both results.

```
if (!empty($string1) && !empty ($string2)) {
  if ($string1 == $string2) {
    echo "<p>Both strings are the same.</p>\n";
  }
  else {
    echo "<p>The strings have " . similar_text ($string1, $string2) .
      " character(s) in common.<br />\n";
    echo "<p>You must change " . levenshtein ($string1, $string2) .
      " character(s) to make the strings the same.<br />\n";
  }
}
else {
  echo "<p>One of the strings does not contain a value so the two
strings cannot be compared.</p>\n";
}
```

6. Open the file in your web browser at the following URL and see the results:

```
http://<server>/<account>/week05/compare.php
```

Exercise 2: Regular Expressions

1. In the `week05` folder, create a partial HTML file with DOCTYPE, html, head, body, meta charset, and title tags. Save the file as `emails.php`.
2. Add a PHP script section to the document body.
3. Use the regular expression from the video to check several different email addresses. The regular expression we used is this. Add this line to your PHP code section.

```
$regex = '/^[A-Za-z0-9\.]+@[A-Za-z0-9]+\.[a-z\.]+$/';
```

4. This regular expression looks for any sequence of letters, numbers, or dots (the user name), then the @ at-sign, then another sequence of letters or numbers (the company name), then a dot, then a sequence of lower case letters (the top-level domain such as com, mil, gov, ca, or uk).
5. Add the code for this array of emails to check:

```
$emails = array (
    "smith@mission.com",
    "jsmith@mission.gov",
    "john.smith@mission.mil",
    "jsmith123@mission.org",
    "john.smith.123@mission",
    "john_smith@mission.com.uk"
);
```

6. Add this loop to check all the emails:

```
for ($i=0; $i<count($emails); $i++) {
    if (preg_match ($regex, $emails[$i]) == 1) {
        echo "GOOD " . $emails[$i] . "<br />\n";
    }
    else {
        echo "BAD  " . $emails[$i] . "<br />\n";
    }
}
```

7. If the code is working properly, the first four emails should get flagged GOOD, and the last two will get flagged BAD. Why? Are the last two emails good or bad? Can you change the regular expression to allow the good email while still rejecting the bad one?
8. Extra credit: Can you use the special escape sequences for digits or word characters to shorten this regular expression?

Exercise 3: Levenshtein vs. Similar_Text

1. In this exercise, we will compare the results of `levenshtein()` vs. `similar_text()` by exercising the functions on an array of strings to find out which are the most similar.
2. In the `week05` folder, create a partial HTML file with DOCTYPE, html, head, body, meta charset, and title tags. Save the file as `similar.php`.
3. Add a PHP script section to the document body.
4. Add this code to the PHP script:

```
$stateNames = array ("Alabama", "Alaska", "California", "Indiana",
                     "Louisiana", "Montana", "Nebraska",
                     "Oklahoma", "Pennsylvania", "Virginia");
```

5. Print all these strings using a loop so we can see what strings you are comparing.
6. Which two of these state names are most similar? We will write code to find out.
7. Write a function to find the smallest possible Levenshtein distance among a set of strings. The Levenshtein distance is the number of changes required to convert one string into the other. So, a smaller Levenshtein distance means the strings are more similar.
8. When trying to find the smallest number, we initialize the variable to a value far greater than the largest value we expect to find. The largest Levenshtein distance in the above array is probably less than 20, because the longest string is less than 20 characters.
9. Add the following function to compute all the Levenshtein distances and return the pair of strings that has the smallest distance.

```
function findLevenshtein($strings, &$word1, &$word2)
{
  $numStrings = count ($strings);
  $smallest = 999; // larger than the largest you expect to find
  for ($i=0; $i<($numStrings-1); $i++) {
    for ($j=$i+1; $j<$numStrings; $j++) {
      $levenshteinValue = levenshtein ($strings[$i], $strings[$j]);
      if ($levenshteinValue < $smallest) {
        $smallest = $levenshteinValue;
        $word1 = $strings[$i];
        $word2 = $strings[$j];
      }
    }
  }
}
```

10. This function takes the list of state names as a first parameter, and it returns the two strings via the second and third parameters, which are pass-by-reference so you can use them to return two separate values. (It is not possible to return two separate values by a return statement.)
11. Call the `findLevenshtein()` function like this:

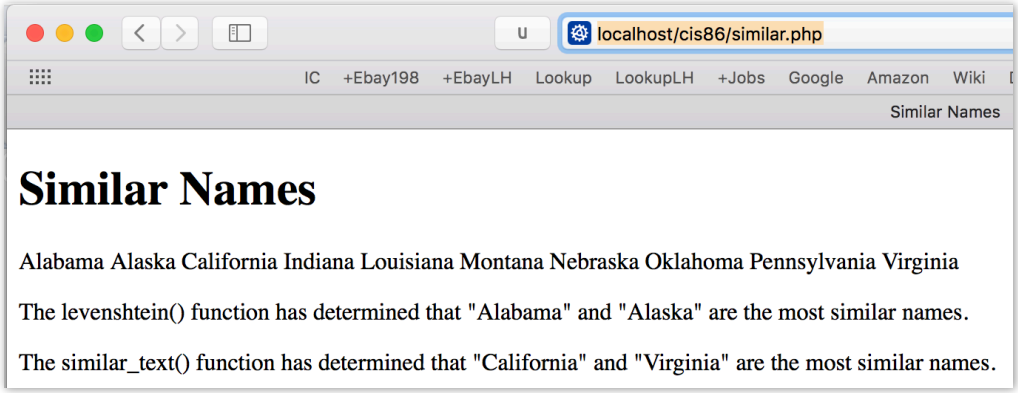
```
findLevenshtein ($stateNames, $lev1, $lev2);
```

Continue on the next page →

12. Print the results using code like this:

```
echo "<p>The levenshtein() function has determined that
  &quot;$lev1&quot; and
  &quot;$lev2&quot; are the most similar names.</p>\n";
```

9. Using the above code as an example, write another function, called `findSimilarText()`, to find the largest value using the `similar_text()` function.
13. The `similar_text()` function computes the number of characters two strings have in common. Therefore, a higher value indicates the strings are more similar. So we are looking for the highest number.
14. When looking for the largest number, we initialize the variable to a value less than the smallest value we expect to find. The smallest possible `similar_text()` result is 0 (zero), because the most dissimilar strings will have zero characters in common. So create a variable `$largest` and initialize it to 0.
15. Compare the strings by calling the `findSimilarText()` function. Write code to print the results.
16. Run the program and view the results:



Exercise 4: Turn it in.

1. Create a folder called `week05` if you have not done so already.
2. Put all the above files in the `week05` folder.
3. Link all the projects in the `week05` folder to your home page. Note that you'll have to use relative links that specify the folder name:

```
<a href='week05/similar.php'>Similar Strings</a>
```

4. Upload your files to the `php.missioncollege.edu` server so I can run them there.
5. Make sure to put the `week05` folder inside the `public_html` folder.