Programming Assignment 4 - More Loops? For Loops!
ECS 10 - Fall 2017

All solutions are to be written using Python 3. Make sure you provide comments including the file name, your name, and the date at the top of the file you submit. Also make sure to include appropriate docstrings for all functions.

The names of your functions must exactly match the names given in this assignment. The order of the parameters in your parameter list must exactly match the order given in this assignment. *All loops in your functions must be for loops*. For any given problem below, you may want to write additional functions other than those specified for your solution. That's fine with us.

Problem 1

Create a Python function called `sumOfOdds` which takes one argument, an integer greater than or equal to 1, and *returns* the result of adding the odd integers between 1 and the value of the given argument (inclusive). This function does not print. Do not use Python's `sum()` function. Do not build a list. You may assume that the argument is valid. Yes, this is *almost* the same question as in the previous programming assignment, except that you must use a for loop instead of a while loop, and you're adding odd numbers instead of even numbers. Here are some examples of how your function should behave:

```
>>> sumOfOdds(1)
1
>>> sumOfOdds(2)
1
>>> sumOfOdds(3)
4
>>> sumOfOdds(5)
9
>>> sumOfOdds(100)
2500
```

Problem 2

Create a Python function called `countChar` which takes two arguments, a single character and a string of arbitrary length, and returns the number of times the character appears in the string. Do not use Python's `count` method. Here are some examples of how your function should behave:

```
>>> countChar("c","abcbdebf")
1
>>> countChar("c","acbcdce")
3
>>> countChar("c","abdefg")
0
>>> countChar("x","")
0
```

Problem 3

Create a Python function called `countDiffs` that takes two arguments, both of which are strings of the same length. (You don't need to verify that the strings are the same length.) Your function should compare the two strings, character by character, and counts the number of times the two strings have different characters in the same location (i.e., at the same integer index). Your function should then return that number. Here are some examples of how your function should behave:

```
>>> countDiffs("abcdef","acceef")
2
>>> countDiffs("abc","abc")
0
>>> countDiffs("abc","xyz")
3
>>> countDiffs("","")
0
```

Problem 4

Create a function called `avgSumOfSquares` that expects one argument, a list of numbers. The function computes the average of the sum of the squares of all the values entered and then returns that value. If the list is empty, the function returns `None`. Do not use `sum()` or `len()`. Here are some examples of how your function should behave:

```
>>> avgSumOfSquares([1,2,3,4,5])
11.0
>>> avgSumOfSquares([3.1, -7.8, 12, 5.5])
61.175
>>> x = avgSumOfSquares([])
>>> print(x)
None
>>> avgSumOfSquares([])
>>>
```

Problem 5

Create a function called `translate` which expects two arguments. The first argument is a string representing a message to be translated into a coded message. The second argument is a list of lists serving as the key to be used to convert the original string to its Morse Code equivalent. Each two-element list in the key represents an alphabetic character that might be found in the original message paired with the Morse Code pattern of dots and dashes that will be substituted for the original character while building the coded message. The function returns a new string created by replacing every character in the message with its counterpart. This function does not print anything. (Hint: you could simplify things by decomposing the `translate` function into smaller functions. Writing `translate` as one big function could get complicated.)

Usually the string passed as the first argument (the message) will include only the characters found as the first elements of the lists within the list that is passed as the second argument (the key). But if something other than one of those characters is found in the message, your function should translate the unexpected character into `"***"` as shown in the example using `test3` below.

```
key = [["a",".-"],["b","-..."],["c","-.-."],["d","-.."],
       ["e","."],["f","..-."],["g","--."],["h","...."],
       ["i",".."],["j",".---"],["k","-.-"],["l",".-.."],
       ["m","--"],["n","-."],["o","---"],["p",".--."],
       ["q","--.-"],["r",".-."],["s","..."],["t","-"],
       ["u","..-"],["v","...-"],["w",".--"],["x","-..-"],
       ["y","-.--"],["z","--.."]]

test1 = "the quick brown fox jumped over the lazy dog"
test2 = "i do not like green eggs and ham"
test3 = "help! i've fallen and I can't get up"
test4 = ""

>>> translate(test1, key)
'- .... .   --.- ..- .. -.-. -.-   -... .-. --- .-- -.   ..-. --- -
..-   .--- ..- -- .--. . -..   --- ...- . .-.   - .... .   .-.. .-
--.. -.--   -.. --- --. '

>>> translate(test2, key)
'..   -.. ---   -. --- -   .-.. .. -.- .   --. .-. . . -.   . --.
--. ...   .- -. -..   .... .- -- '

>>> translate(test3, key)
'.... . .-.. .--. ***   .. *** ...- .   ..-. .- .-.. .-.. . -.   .-
-. -..   ***   -.-. .- -. *** -   --. . -   ..- .--. '

>>> translate(test4, key)
''
```

As in the examples above, the returned string should have exactly one space between the Morse code patterns within a single word (i.e., `"the"` becomes `'- .... .'`) and exactly three spaces between words
(i.e., `"the lazy"` becomes `'- .... .   .-.. .- --.. -.--'`).

Where to do the assignment

You can do this assignment on your own computer, or in the labs. In either case, use the IDLE development environment -- that's what we'll use when we grade your program. Put all the functions you created in a file called "prog4.py".

Submitting the Assignment

Go to Canvas and submit the file containing your functions as an attachment. Do NOT cut-and-paste your functions into a text window. Do NOT hand in a screenshot of your functions' output. We want one file from you: "prog4.py".

Saving your work

If you are working in the lab, you will need to copy your program to your own flash-drive. To save it on flash-drive, plug the flash-drive into the computer (your TA or the staff in the labs can help you figure out how), open the flash-drive, and copy your work to it by moving the folder with your files from the Desktop onto the flash-drive.