CSCI 1100 — Computer Science 1 Homework 4
Loops and Lists

Overview

This homework is worth **90 points** total toward your overall homework grade (each part is 30 points), and is due Thursday, October 15, 2015 at 11:59:59 pm. There are three parts to the homework, each to be submitted separately. All parts should be submitted by the deadline or your program will be considered late.

Note on grading. As mentioned in past homeworks, correctness and program structure will both be a part of your grade. Use functions, put at least one top comment, name your variables properly and structure your code to put functions definitions before any program code. To get you started, we have also included one example program that you must use as a starting point for part 1 in the zip file for this homework. The homework submission server URL is below for your convenience:

```
https://submit.cs.rpi.edu/index.php?course=csci1100
```

Your programs for each part for this homework should be named:

```
hw4_part1.py
hw4_part2.py
hw4_part3.py
```

respectively. Each should be submitted separately.

Part 1: Words with alternating vowel and consonant

Write a program that reads in a word entered by the user and checks whether:

- the word has at least 8 characters,
- starts with a vowel,
- has alternating vowels and consonants, and
- the consonants are in increasing alphabetical order.

Note that you can check letters for alphabetical order using `<`.

For example:

```
>>> 'a' < 'b'
True
>>> 'z' < 'b'
False
```

Your program should work for words entered upper or lower case, and must use a single function `is_alternating(word)` that returns `True` if the word has the above pattern, and `False` otherwise. (**Hint.** Remember to write a loop that goes through each letter and check for the necessary conditions. Using indexing is important here as you need to compare letters in different positions. We have looked at a similar piece of code in our first lecture!) Here is an example run of this program:

```
Enter a word => eLimiNate
eLimiNate
The word 'eliminate' is alternating
```

```
Enter a word => ageneses
ageneses
The word 'ageneses' is not alternating
```

```
Enter a word => ADAGE
ADAGE
The word 'adage' is not alternating
```

The word `eliminate` has this alternating pattern since we have that: `'l'<'m'<'n'<'t'`.

The word `adage` is not long enough, `ageneses` has two consonants that are identical and as a result not in strictly increasing alphabetical ordering.

When you have tested your code, please submit it as `hw4_part1.py`.

Part 2: Legos

In this part, we will work with legos. Suppose you are given a list of lego pieces that you own, but you have a new project. You want to see if you have enough of a specific piece. Even if you do not have the exact piece, you can put together different lego pieces to make up bigger pieces.

Figure 1: Left: all the possible lego pieces for this homework, name is dimension of the lego (no 1x2, only 2x1). Right: example of combining two 2x1 pieces to make up a 2x2 piece.

Write a program to read from a file the list of all lego pieces you currently have. Ask the user for the type of lego piece and the number of such pieces that she is searching for. Then, return whether you can make that many pieces of legos by using the legos in your collection. You will only consider methods in which one type of lego is used. For example, you can make up a `2x2` lego using: two `2x1` legos, or four `1x1` legos. Also, return the list of remaining legos after this request is satisfied.

Here are some sample outputs of your program:

```
Current legos ['1x1', '1x1', '1x1', '1x1', '1x1', '1x1', '2x1', '2x1', '2x1', '2x1',
    '2x2']
Type of lego wanted => 2x1
2x1
Quantity wanted => 2
2
I can use 2 2x1 legos for this
Remaining legos ['1x1', '1x1', '1x1', '1x1', '1x1', '1x1', '2x1', '2x1', '2x2']
```

```
Current legos ['1x1', '1x1', '1x1', '1x1', '1x1', '1x1', '2x1', '2x1', '2x1', '2x1',
    '2x2']
Type of lego wanted => 2x2
2x2
Quantity wanted => 2
2
I can use 4 2x1 legos for this
Remaining legos ['1x1', '1x1', '1x1', '1x1', '1x1', '1x1', '2x2']
```

```
Current legos ['1x1', '1x1', '1x1', '1x1', '1x1', '1x1', '2x1', '2x1', '2x1', '2x1',
    '2x2']
Type of lego wanted => 2x1
2x1
Quantity wanted => 6
6
I don't have 6 2x1 legos
```

In the first example, we have exact match. In the second case, we use substitution. In the third example, we search for 6 `2x1` legos or 12 `1x1` legos. Given we do not have either one, we fail. Note that a different solution exists to this last case using 4 `2x1` and 4 `1x1` legos. This is a much harder problem (mix and match) and we will not allow it for simplicity.

To solve this problem, you will first read from a file how many of each type of lego pieces you currently have, such as the one below:

```
6 1x1
4 2x1
1 2x2
```

using the function we provided you in `hw4_util` as follows:

```
import hw4_util
legos = hw4_util.read_legos()
print legos
```

If you execute this program with the above file, you will get the list below.

```
['1x1', '1x1', '1x1', '1x1', '1x1', '1x1', '2x1', '2x1', '2x1', '2x1', '2x2']
```

A very easy way to solve this problem is to use the `count()` function of lists. For example, given the above list, `legos.count('1x1')` returns 6. You need to write the if statements to check for each lego, whether a substitute exists. You also need to use list functions to remove the found legos from the list of existing legos. Always try to using the biggest possible lego first.

It should be obvious how you can put smaller pieces together to make up bigger pieces, but we provide possible subtitutions here for completeness. We only use one type of lego for any request, no mix and match.

Piece	Possible replacement
2x1	2 1x1
2x2	2 2x1
	4 1x1

Note that we only gave you one test file. But, you must create other test files to make sure that the program works for all possible cases. Feel free to share your test files on Piazza and test cases. Discussing test cases on Piazza are a good way to understand the problem.

We will test your code with different input files than the one we gave you in the submission server. So, be ready to be tested thoroughly!

Ask us for suggestions on how to organize your code on Piazza, we will provide suggestions. This is a good homework to test how to input a function as an argument to a list and modify the list in that function (for removal of legos). I also recommend you separate the search for substitutions in a different function, for a simpler program.

When you have tested your code, please submit it as `hw4_part2.py`.

Part 3: Google Flu Trends

Have you heard about Google Flu Trends? Researchers have shown that when the flu season starts, the Google queries on flu symptoms and treatments goes up significantly. So, we can look at Google queries to see what is happening in the world. In this homework, you will be using real Google flu trend data for 2014 for many different countries.

Write a program to read in the flu trends for different countries for all of the 52 weeks in 2014. Print the trend in a single line. For each week in which the flu search was higher than usual, print a plus sign and otherwise print a minus sign. Also print a header for the months of the year, starting with 1. Here is a possible run:

```
First country => US
US
Second country => Switzerland
Switzerland
            1   2   3   4   5   6   7   8   9   10  11  12
US          ----++++++----------------------++++++----------
Switzerland ---++++++-----------------------------------------
```

The country name is printed as entered by the user left justified to 12 characters. Each month is printed left justified 4 characters.

To accomplish this, you will use another function from the utility module provided for this homework. Given a string containing a country name, it returns you a list of 52 numbers. Try the following:

```
import hw4_util
cdata = hw4_util.read_flu('Switzerland')
print cdata
```

You will get the following list:

```
  [5, 6, 7, 12, 15, 18, 19, 12, 9, 9, 8, 6, 5, 6, 4, 5, 3, 3, 3, 3, 2,
```

```
2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 4, 4, 4, 3, 3, 4, 4, 4, 4, 4,
5, 4, 4, 5, 5, 6, 6, 4, 7]
```

You have 52 values corresponding to the flu based search frequency for each week. Apparently the Swiss really don't really search for flu online.

Now, given a country such as this, you can find when flu is peaking for this country. It is when the flu volume is higher than or equal to the cutoff value which we will compute as:

`cutoff = (average value + max value)/2`

For the above country, cutoff would be 12 (average is 5 and max is 19, so the cutoff is 12 using integer division.)

We only have five values higher than cutoff for the above list. We will now print a line for this country by printing a minus sign when the value is below cutoff and a plus sign otherwise, as shown below:

`Switzerland ---+++++-----------------------------------------`

The function `hw4_util.read_flu` will return an empty list whenever the country cannot be found. Your program must give an error message for all missing countries as shown below.

```
First country => Kanada
Kanada
Second country => Meksiko
Meksiko
I could not find country Kanada
I could not find country Meksiko
```

```
First country => US
US
Second country => Kanada
Kanada
I could not find country Kanada
```

When you have tested your code, please submit it as `hw4_part3.py`.