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, March 10, 2016 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.

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.

See the handout for Homework 3 for discussion of grading and for a discussion of what is considered excess collaboration. These rules will be in force for the rest of the semester.

Part 1: Palindromes

Write a program that reads in a word or sentence entered by the user and checks whether the word is a palindrome (reads the same forward and backward):

- ignore capitalization;
- ignore all non-alphabetic characters including punctuation and digits;
- if the word or sentence is a palindrome, print a message; and
- if the word or sentence is not a palindrome, print the characters which failed the palindrome test and indicate whether the original or reversed string comes first in alphabetic order.

Your program should work for words entered in upper or lower case. Write at least one function, `remove_extra(word)`, that takes a string; removes all whitespace and non-alphabetic characters; changes to lower case; and returns the modified string.

```
Enter a word or a sentence => Madam, I'm Adam!
Madam, I'm Adam!

String after removing non-alphabetic characters: madamimadam
String reversed: madamimadam
Palindrome
```

```
Enter a word or a sentence => Madam, am I Adam?
Madam, am I Adam?

String after removing non-alphabetic characters: madamamiadam
String reversed: madaimamadam
Not a palindrome: m comes after i at position 4
```

```
Enter a word or a sentence => Mada I'm Adam!
Mada I'm Adam!

String after removing non-alphabetic characters: madaimadam
String reversed: madamiadam
Not a palindrome: i comes before m at position 4
```

```
Enter a word or a sentence => Madam, I'm 1-Adam-12!
Madam, I'm 1-Adam-12!

String after removing non-alphabetic characters: madamimadam
String reversed: madamimadam
Palindrome
```

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.

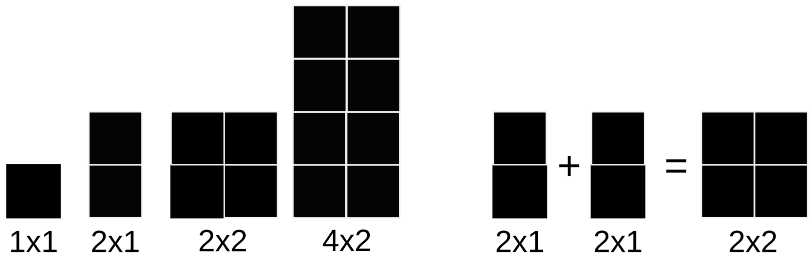

Figure 1: Left: all the possible lego pieces for this homework, name is dimension of the lego (no 1x2, only 2x1). Right: example of combining two 2x1 pieces to make up a 2x2 piece.

Write a program that starts by reading from a file — using a module we have provided – a list of lego pieces that you currently have. Then, the program should ask for the type of lego piece you want to make and for the number of that type of piece you need. The program should then tell you whether you can make that many lego pieces using the legos in your collection. The program should also print the number of remaining lego pieces you will have available after this request is satisfied.

The program should **only consider methods in which one type of lego is used to make the desired lego**. For example, you can make up a `2x2` lego using one `2x2` lego (obviously), two `2x1` legos, or four `1x1` legos. You can make up a `4x2` lego using one `4x2` lego, two `2x2` legos, four `2x1` legos, or eight `1x1` legos. If there are multiple solutions, your program should always choose the solution the uses the fewest number of pieces (i.e. the large piece). Solutions that combine different sized pieces should not be considered.

Here are some sample outputs of your program:

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

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

```
Current legos ['1x1', '1x1', '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
```

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

In the first example, we have an exact match. In the second example, 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. In the fourth example, we have two possible solutions, four `2x1` legos or eight `1x1` legos. The program chooses four `2x1` legos as the solution because it uses fewer pieces.

The file `legos.txt` contains a list of your available legos. The format of the file is shown below:

```
8 1x1
4 2x1
1 2x2
0 4x2
```

We provide a function to read the file format in `hw4_util`. You can use it as follows:

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

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

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

Remember your list functions. Using the `count()` function of lists will make this program much easier. For example, given the above list, `legos.count('1x1')` returns 8. You will need to write the input code, the control structures, (if and/or while statements), and other code needed to check for each lego and to remove the required legos from the list of existing legos. Again, always use 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 substitutions here for completeness. We only use one type of lego for any request, there is no mix and match.

Piece	Possible replacement
2x1	2 1x1
2x2	2 2x1
	4 1x1
4x2	2 2x2
	4 2x1
	8 1x1

Note that we only gave you the one test file. You must edit this file to create other starting configurations for your legos. Make sure that the program works for all possible cases. We will definitely test your code with different input files than the one we gave you in Piazza. For this homework, feel free to share your test files and test cases on Piazza, but remember that your code must be your own. **Do not share code!**

This is a good homework to test how to input a function as an argument to a list and to modify the list in that function (for removal of legos). We recommend you separate the search for substitutions in a different function. It will make your program simpler and simple code is good code.

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

Part 3: New York State Death Statistics

As part of an open government inititive, New York State releases a large number of health related statistics for researchers and developers to use. Among these are death statistics by region of the state at `https://health.data.ny.gov`. For this assignment, we have simplified the statistics to provide just total number of deaths per 100,000 people. Our goal is to come up with a simple visualization summarizing death trends over the last few years.

Write a program to read in the death statistics for different areas of NYS for the 11 years from 2003 to 2013. Print the trend in a single line. For each year where the death trend was at least 5 percent above the average for the county print a plus sign. For each year where the death trend was at least 5 percent below the average for the county print a minus sign. Otherwise, print an equal sign. Also print a header for the data labeling the start and end years. Note that this should be in reverse order from 2013 to 2003.

```
First county => Allegany
Allegany
Second county => Cattaraugus
Cattaraugus

               2013   2003
Allegany       =++-=-=-===
Cattaraugus    ===+=====+=
```

The county name is printed as entered by the user left justified to 15 characters. 2013 and 2003 are printed to line up with the margins of the trend data.

The utility module provided for this homework will give you some help. Given a string containing a county name, it provides a function `read_deaths(county)` that returns you a list of 11 numbers. Try the following:

```
import hw4_util
cdata = hw4_util.read_deaths('Allegany')
print cdata
```

You will get the following list:

```
[958.2, 978.1, 994.3, 852.0, 897.1, 880.2, 961.0, 875.3, 950.6, 897.7, 957.2]
```

You have 11 values corresponding to the total deaths per 100,000 people during the years from 2013 to 2003. For the above county, the high cutoff would be 973.80 and the low cutoff would be 881.06 (average is 927.43) We only have two values higher than the high cutoff and three values lower than the low cutoff. We will now print a line for this county by printing a plus sign when the value is above the high cutoff, a minus sign when it is below the low cutoff, and an equals sign otherwise, as shown below:

```
Allegany       =++-=-=-===
```

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

```
First county => Errie
Errie
Second county => Cataraugis
Cataraugis

I could not find county Errie
I could not find county Cataraugis
```

```
First county => Allegany
Allegany
Second county => WESStCHester
WESStCHester

I could not find county WESStCHester
```

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