CSCI 1100 — Computer Science 1 Homework 3
Lists and If statements

Overview

This homework is worth **90 points** toward your overall homework grade, and is due Thursday, October 2, 2014 at 11:59:59 pm. In the zip folder associated with this homework you will find `hw3_util.py`, `team_scores.txt`, and `legos.txt`. The first is a module that is written to help you read information from files. You should be able to write this module in about a week yourself. But, for now, you can simply use it.

The others are text files that you will use in different parts of this homework.

The goal of this assignment is to work with lists and use if statements. As your programs get longer, you will need to develop some strategies for testing your program. Start early, test small parts of your program by writing a little bit and testing. We will walk you through program construction in the homework description and provide some ideas for testing.

As always, make sure you follow the program structure guidelines. You will be graded on program correctness as well as good program structure.

Fair Warning About Excess Collaboration

For HW 3 and for the remaining homework assignments this semester we will be using software that compares **all** submitted programs, looking for inappropriate similarities. This handles a wide variety of differences between programs, so that if you either (a) took someone else's program, modified it (or not), and submitted it as your own, or (b) wrote a single program with one or more colleagues and submitted modified versions separately as your own work, this software will mark these submissions as very similar. Both (a) and (b) are beyond what is acceptable in this course — they are violations of the academic integrity policy. Furthermore, this type of copying will prevent you from learning how to solve problems and hurt you in the long run. The more you write your own code, the more you learn.

Here are some strategies for collaborating in the homework and learning from it:

- It is perfectly acceptable to work on a solution with a friend, debug each others' code, etc. However, when you are finished, you must write your own code, from scratch. You must not copy from a solution you worked with someone. This is a test for yourself. If faced with the blank page, can you write a program? Also remember, repetition is how you memorize the basic patterns. If you write lots of code, you will see common patterns and start to abstract from them. That is what programming is all about.
- When working with TAs, mentors, tutors or any other helpful people who seem to materialize when you are writing programs, ask them to give you an example on a piece of paper or their own computer. Then, write it yourself without looking at it. You are cheating yourself if someone stands behind you and tells you what to write. You are not learning to code if you do this, you are just a typist. Challenge yourself continuously.
- Unfortunately, you must also consider that not everyone is well meaning. Do not leave your code around. Do not give your code to people who will not follow the code of conduct that we just described. If they copy your code and turn it in, you will be in trouble too.

To protect those of you who follow these rules of conduct, we will monitor similarities between programs. Students whose programs are quite similar will be given a chance to explain their source of similarities. The standard penalty for submitting work that is not your own, or for providing code that another student submits (perhaps after modification) will be

- 0 on the homework, and
- an additional overall 5% reduction on the semester grade.

Penalized students will also be prevented from dropping the course. More severe violations, such stealing someone else's code, will lead to an automatic F in the course. A student caught in a second academic integrity violation will receive an automatic F.

By submitting your homework you are asserting that you both (a) understand the academic integrity policy and (b) have not violated it.

Finally, please note that this policy is in place for the small percentage of problems that will arise in this course. Students who follow the strategies outlined above and use common sense in doing so will not have any trouble with academic integrity.

Part 1: FIFA world cup, and its crazy rules

In this homework, we will be making use of a module called `hw3_util.py` that provides a function for this part of the homework. To get started with this part, unzip the folder given to you containing the input files and this util file. You will start with reading the file containing information about teams in FIFA world cup, the group round. Try the following:

```
import hw3_util
teams = hw3_util.read_fifa()
print teams[0]
```

The function called `read_fifa()` will return you a list of teams. Each item in the list is information for a team given in a list containing the following information:

`[group id, country, games, win, draw, lose, goals scored, goals against]`

Write a program to compare any two teams from this list and to print which team is superior. Here are the rules:

- The team with the highest points is superior. Points are calculated as follows: each win is 3 points and each draw is 1 points.
- If the points are the same, then the team with the higher goal difference is superior. The goal difference is compuated as follows: subtract the number of goals against from the number of goals scored.
- If the two teams have the same goal difference, then the team with the higher number of goals scored is superior.
- If the two teams are the same with respect to all of the above rules, we will assume that they are the same. In this case, print that the two teams are the same. FIFA has even rules for breaking the ties for that. But, we will stop here.

Your program must ask the index of any two teams from the list. Assume a valid index is given. Then, it will first print the full information corresponding to these teams. The format is given below (the country column is 20 characters long, all other columns are 6 characters long).

You will then compare the two teams corresponding to the input indices, and print which team is greater. Here are some possible runs of the program:

```
First team index (0-31) ==> 2
2
Second team index (0-31) ==> 3
3

Group Team                Win   Draw  Lose  GF    GA    Gdiff Pts
1     Croatia             1     0     2     6     6     0     3
1     Mexico              2     1     0     4     1     3     7
Mexico is better than Croatia


First team index (0-31) ==> 0
0
Second team index (0-31) ==> 3
3

Group Team                Win   Draw  Lose  GF    GA    Gdiff Pts
1     Brazil              2     1     0     7     2     5     7
1     Mexico              2     1     0     4     1     3     7
Brazil is better than Mexico


First team index (0-31) ==> 10
10
Second team index (0-31) ==> 14
14

Group Team                Win   Draw  Lose  GF    GA    Gdiff Pts
3     Ivory Coast         1     0     2     4     5     -1    3
4     Italy               1     0     2     2     3     -1    3
Ivory Coast is better than Italy
```

Remember to print any value we read to make the output appear correctly at the submission server.

To match the formatting, you can use the padding function `ljust()` that will pad the input string with spaces on the right up to the given length.

```
>>> x = "abc"
>>> x.ljust(6)
'abc   '
```

Once you have tested your program, turn in only your code in a file named `hw3_part1.py`.

Part 2: Legos (Everything is awesome!)

In celebration of everyone's childhood, we have a lego problem in this homework. We will solve a simple problem in this part. But, then we will revisit the problem and solve a harder version in a future homework.

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. But, you can put together different lego pieces to make up bigger pieces too.

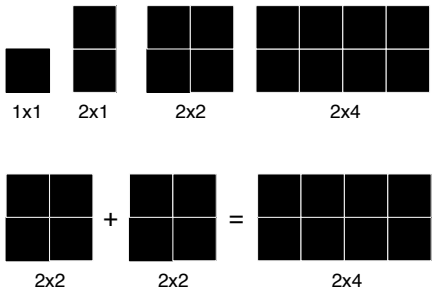

Figure 1: All the possible lego pieces for this homework are shown on the top row. The name explains the dimensions of the lego. The bottom row shows how you can combine two 2x2 pieces to make up a 2x4 piece.

Write a program to read from a file the list of all lego pieces you currently have. Then, 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 repeated. For example, you can make up a `2x4` lego using: two `2x2` legos, or four `2x1` legos or eight `1x1` legos.

Here are some sample outputs of your program:

```
What type of lego do you need? ==> 2x4
2x4
How many pieces of this lego do you need? ==> 1
1
I have 1 pieces of 2x4 for this


What type of lego do you need? ==> 2x1
2x1
How many pieces of this lego do you need? ==> 2
2
I have 2 pieces of 2x1 for this


What type of lego do you need? ==> 2x1
twobyone
How many pieces of this lego do you need? ==> 3
3
I have 6 pieces of 1x1 for this
```

```
What type of lego do you need? ==> 2x1
2x1
How many pieces of this lego do you need? ==> 4
4
I don't have enough pieces of this lego
```

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:

```
1x1, 6
2x1, 2
2x2, 2
2x4, 1
```

using the function provided in `hw3_util` as follows:

```
import hw3_util
legos = hw3_util.read_legos('legos.txt')
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', '2x2', '2x2', '2x4']
```

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 4. You need to write the if statements to check for each lego, whether a substitute exists. 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
2x4	2 2x2
	4 2x1
	8 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!

Once you are done, turn in only your code `hw3_part2.py`.