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 6, 2016 at 11:59:59 pm. In the zip folder associated with this homework you will find `read_names.py`, `names_example.py`, `top_names_1880_to_2014.txt`, `hw3_util.py`, and `legos.txt`. The modules `read_names.py` and `hw3_util.py` are written to help you read information from files. You should be able to write these modules in about a week yourself. But, for now, you can simply use them.

The others are examples you can look at, and 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. Here are a few simple ones: start testing early, and 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, (b) wrote a single program with one or more colleagues and submitted modified versions separately as your own work, or (c) submitted (perhaps slightly modified) software submitted in a previous year as your software, this software will mark these submissions as very similar. All of (a), (b), and (c) 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 as 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: How Have the Popular Baby Names Changed?

Create a folder for HW 3. Download the zip file `hw3Files.zip` from Piazza. Put it in this folder and unzip it. You should see

```
read_names.py
names_example.py
top_names_1880_to_2014.txt
hw3_util.py
legos.txt
```

The first three will be used in this part, while part 2 will use `hw3_util` and `legos.txt`. Do all of your work for part 1 in this folder. Start by opening `names_example.py` in the Wing IDE and then read the following...

We have data from the Social Security Administration that gives the top 250 female and male baby names for every year from 1880 up to and including 2014, and also gives you how many babies were given each name. These data are stored the file `top_names_1880_to_2014.txt`.

We have also provided you with a module called `read_names.py` that gives you easy access to this data. The program `names_example.py` provided to you and shown here illustrates the use of this module and the resulting lists:

```
'''
Initial example to demonstrate how to read and access the baby names
and counts.
'''
import read_names

# Read in all the names.  The result is stored in the module.
read_names.read_from_file("top_names_1880_to_2014.txt")

# Access the female names and counts for 1886
(female_names, female_counts) = read_names.top_in_year(1886, 'f')
print("The most common female name in 1886, with a count of {:d}, is {:s}"\
    .format(female_counts[0], female_names[0]))

# Access the male names and counts for 1997.  Note that the 100th most
# popular name is in position 99 of the list.
(male_names, male_counts) = read_names.top_in_year(1997, 'M')
print("The 100th most common male name in 1997, with a count of {:d}, is {:s}"\
    .format(male_counts[99], male_names[99]))
```

Play around with code until you understand what it is doing and how to work with the two lists that are returned. Only when you are comfortable should you proceed to the actual assignment...

Write a program that does the following:

1. It asks the user for a `year` in the range from 1880 up to and including 2014. It should check to see if the year is in the appropriate range and if it is not the program should print an error message (see example below) and stop.
2. It asks the user for a female name and finds the index of the name in the list.
3. It then finds the index of the name at plus and minus 5 and 10 years from the selected date, ignoring any dates that are outside of the 1880 - 2014 range.

4. At this point, for each valid year in the test set `year-10`, `year-5`, `year`, `year+5`, `year+10`, if the name was not found in that year the program should print a message saying it was not found. If the name was found in that year it outputs the statistics about the name including the year tested, the rank, the count, the percentage of the count relative to the count of the top ranked name, and the percentage of the name relative to the sum of all the name counts in the top 250. For any index less than 0 or greater than 249, nothing should be printed.

5. It should ask for a male name and do the same.

Example output when running from the Wing IDE is below. Formatting should use 3 spaces for the rank `{:3d}`, 5 spaces for the count (`{:5d}`), and 7 spaces for each percentage (`{:7.3f}`).

First example

```
Enter the year to check => 2014
2014
Enter a female name => Emma
Emma
Data about female names
Emma:
    2004:   2 21599  86.310   2.145
    2009:   2 17881  80.263   1.925
    2014:   1 20799 100.000   2.321

Enter a male name => Jonathan
Jonathan
Data about male names
Jonathan:
    2004:  21 14357  51.512   1.034
    2009:  29 11359  53.722   0.877
    2014:  44  8035  41.971   0.665
```

Second example

```
Enter the year to check => 1952
1952
Enter a female name => Mary
Mary
Data about female names
Mary:
    1942:   1 63238 100.000   5.587
    1947:   2 71684  71.914   4.807
    1952:   2 65681  97.891   4.238
    1957:   1 61096 100.000   3.657
    1962:   2 43497  94.388   2.784

Enter a male name => Jackson
```

```
Jackson
Data about male names
Jackson:
    1942: Not in the top 250
    1947: Not in the top 250
    1952: Not in the top 250
    1957: Not in the top 250
    1962: Not in the top 250
```

Third example

```
Enter the year to check => 2015
2015
Year must be at least 1880 and at most 2014
```

Here are several notes to help you.

- If you import a module called `sys` then you can stop your program from executing with the statement `sys.exit()`. This is a nice thing to be able to do when you find an error in your input.
- You can easily pass lists as arguments to functions. For example

```
def total_and_avg( one_list ):
    n = len(one_list)
    total = sum(one_list)
    avg = total/float(n)
    print('first {:d}, total {:d}, avg {:5.2f}'.format(one_list[0], total, avg

y = [ 8, 2, 4, 5 ]
total_and_avg( y )
```

- Writing one or two functions can make the code much, much simpler. One function might be given as parameters a single name, the year, the list of names and the list of counts, and from these it can print a line of the table. The function would be called once for each year and would ignore years outside of the valid range. A second function, called from within this first function, might calculate and print the statistics for a given index in the counts list, or perhaps return the statistics as a tuple.
- The following example will help you understand how to find if a value is in the list and the list index.

```
>>> a = [165, 32, 89, 16 ]
>>> x = 32
>>> x in a
True
>>> i = a.index(x)
>>> i
```

```
1
>>> print(a[i], a[i+1])
32 89
>>> 183 in a
False
>>> j = a.index(183)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 183 is not in list
```

Note that it is an error to ask for an index that is not in the list!

When you are finished, submit your program to Submitty as `hw3Part1.py`. You must use this filename, or your submission will not work in Submitty. You do **not** have to submit any of the files we have provided. Note that you do not need loops for this part. Use the appropriate list functions such as `sum` and `find` to generate the answers. Use of loops will be penalized in the grading.

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.

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.

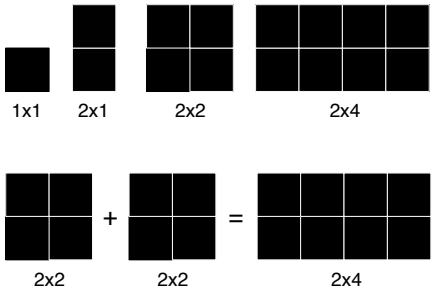

Figure 1: All the possible lego pieces for this homework are shown on the top row. The name explains the dimensions of the lego. The bottom row shows how you can combine two 2x2 pieces to make up a 2x4 piece.

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 that she is searching for. If the lego type is not in the list of possible pieces, print out `Illegal lego` and exit Otherwise, print out how many of that piece you can make 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. In other words, you do not consider the possibility of using two `2x1` legos and four `1x1` legos.

Let's assume you have the following lego pieces available to you:

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

Here are some sample outputs of your program:

```
What type of lego do you need? ==> 2x4
2x4

I can make 2 2x4 pieces:
     1 pieces of 2x4 using 2x4 pieces.
     1 pieces of 2x4 using 2x2 pieces.
     0 pieces of 2x4 using 2x1 pieces.
     0 pieces of 2x4 using 1x1 pieces.

What type of lego do you need? ==> 2x1
2x1

I can make 5 2x1 pieces:
     0 pieces of 2x1 using 2x4 pieces.
     0 pieces of 2x1 using 2x2 pieces.
     2 pieces of 2x1 using 2x1 pieces.
     3 pieces of 2x1 using 1x1 pieces.

What type of lego do you need? ==> 4x2
4x2

Illegal lego
```

To solve this problem, you will first read from a file how many of each type of lego pieces you currently have 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 code with the above file, you will get the `legos` 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 6. You need to write the if statements to check for each lego, whether a substitute exists.

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, 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!

When you are finished, submit your program to Submitty as `hw3Part2.py`. You must use this filename, or your submission will not work in Submitty. You do **not** have to submit any of the files we have provided. Note that you do not need loops for this part. Use the appropriate list functions. Use of loops will be penalized in the grading.

As an important final note: It is easy to overcomplicate this problem by doing calculations based on the sizes of the legos. You do none of this!! You should write your code assuming that the **only** legos that can possibly exist are `'1x1'`, `'2x1'`, `'2x2'` and `'2x4'` and you should hard-code all of the replacements outlined above.

Part 3: Turtle moves!

Suppose you have a Turtle that is standing in the middle of an image, at coordinates (200,200) facing right (along the dimensions). Assume top left corner of the board is (0,0) like in the images.

You are going to read input from the user five times. Though you do not have to use a loop for this part, it will considerably shorten your code to use one. It is highly recommended that you use a loop here. However, using a loop in place of simple list functions such as sorting will incur penalties during grading.

Each user input will be a command to the Turtle. You are going to execute each command as it is entered, print the Turtle's current location and direction. At the same time, you are going to put all the given commands in a list, sort this resulting list at the end of the program and print it as is (no other formatting).

Here are the allowed commands:

- `move` will move Turtle 20 steps in its current direction.
- `jump` will move Turtle 50 steps in its current direction.
- `turn` will turn the Turtle 90 degrees counterclockwise: right, up, left, down.
- `sleep` will keep the turtle in the same spot for **two** turns.

If user enters an incorrect command, you will do nothing and skip that command. You should accept commands in a case insensitive way (move, MOVE, mOve should all work).
There is a fence along the boundary of the image. No coordinate can be less than 0 or greater than 400. 0 and 400 are allowed. We realize that not all boundaries can be reached in 5 turns, but check it anyway. It will make the program less fragile should we make the number of turns variable in a later homework.
You must implement two functions for this program:

- `move(x,y,direction,amount)`

  will return the next location of the Turtle as an `(x,y)` tuple if it is currently at `(x,y)`, facing `direction` (direction one of `right, up, left, down`) and moves the given amount.

- `turn(direction)`

  will return the next direction for the Turtle currently facing `direction`, moving counterclockwise.

Now, write some code that will call these functions for each command entered and update the location of the Turtle accordingly. Here is an example run of the program:

```
Turtle: (200, 200) facing: right
Command (move,jump,turn,sleep) => turrn
turrn
Turtle: (200, 200) facing: right
Command (move,jump,turn,sleep) => turn
turn
Turtle: (200, 200) facing: up
Command (move,jump,turn,sleep) => Move
Move
Turtle: (200, 180) facing: up
Command (move,jump,turn,sleep) => turn
turn
Turtle: (200, 180) facing: left
Command (move,jump,turn,sleep) => jumP
jumP
Turtle: (150, 180) facing: left

All commands entered: ['turrn', 'turn', 'Move', 'turn', 'jumP']
Sorted commands: ['Move', 'jumP', 'turn', 'turn', 'turrn']
```

In the above example, `turrn` is an invalid command, so it has no effect in the Turtle's state. In case you are wondering, the list is sorted by string ordering, which is called lexicographic (or dictionary) ordering.

Here is a second example:

```
Turtle: (200, 200) facing: right
Command (move,jump,turn,sleep) => sleep
sleep
Turtle falls asleep.
Turtle: (200, 200) facing: right
Turtle is currently sleeping ... no command this turn.
Turtle: (200, 200) facing: right
Command (move,jump,turn,sleep) => Move
Move
Turtle: (220, 200) facing: right
Command (move,jump,turn,sleep) => turn
turn
Turtle: (220, 200) facing: up
Command (move,jump,turn,sleep) => sleep
sleep
Turtle falls asleep.
Turtle: (220, 200) facing: up

All commands entered: ['sleep', 'Move', 'turn', 'sleep']
Sorted commands: ['Move', 'sleep', 'sleep', 'turn']
```

When you are finished, submit your program to Submitty as `hw3Part3.py`. You must use this filename, or your submission will not work in Submitty.