CPSC 217: Introduction to Computer Science for Multidisciplinary Studies I

Assignment 4: Zombie Contact Tracing

Weight: 8%

Collaboration

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, what you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:

   ```
   # The following code is from
   # https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```

   Use the complete URL so that the marker can check the source.

2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still receive a low grade if you submit code that is not primarily developed yourself. Cited material should never be used to complete core assignment specifications.**
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's screen, then this code is not your own.
4. **Collaborative coding is strictly prohibited**. **Your assignment submission must be strictly your own code**. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
5. Making your code available, even passively, for others to copy, or potentially copy, is also plagiarism.

6. We will be looking for plagiarism, possibly using automated software designed for the task. For example, one tool that we may use is Measures of Software Similarity (MOSS). Information about MOSS can be found here: https://theory.stanford.edu/~aiken/moss/.
7. Remember, if you are having trouble with an assignment, it is always better to ask your TA and/or instructor for help than it is to plagiarize. A common penalty for plagiarism is an F on the plagiarized assignment.

Late Penalty

Late assignments will not be accepted.

Goal

Write a well-structured program that includes dictionaries, command line arguments, file I/O, and exception handling.

Technology

Python 3

Submission Instructions

You must submit your assignment electronically. Use the Assignment 4 drop box in D2L for the electronic submission. You can submit multiple times, with each subsequent submission overwriting any previous submission(s). Do not wait until the last minute to attempt to submit. Your work will not be graded if you attempt to submit at the last minute and time runs out before you are able to complete the process. Your assignment must be completed in **Python 3** and be executable with Python version 3.6.8 or greater. Do not import any additional libraries to complete the assignment other than the formatList module referenced in Part 1.

Description

In this assignment you will analyze fictious contact tracing data. This contact tracing data is one-directional, meaning people who have tested positive for a fictitious zombie disease will report the people with which they have had contact. We won't have data for the reverse direction where a well person had contact with a sick individual. In order to work with the contract tracing data, you will need to load an indicated data file storing contact tracing data. You will then store this data in **dictionary** and **list** data structures. These data structures will then be used to identify the relationship between the individuals in the data.

Each of parts 2 through 7 should be implemented as a function. For parts 2 through 6 the function returns a list. In Part 7 the function returns a dictionary. Each function should examine a dictionary input parameter, which holds the input file data, and then return the information that fulfills the requirements of that part of the assignment. You program will then **print the information returned using print() commands in main()**.

The reading of arguments, file handling, the Part 1 code, and the printing for the remaining parts can all happen in a single main function. You are welcome to use more functions if you desire, but you should have at least this minimal structure.

File Format

The information is stored in CSV (comma separated value) files. Example files (as well as your assignment's expected output for each file) are posted on the course website. Each file ends with the .txt extension. (*Typically, CSV files end with the .csv extension but many operating systems, by default, open these files in a spreadsheet program like Excel and when saving them will change their contents in a way that disrupts this assignment. Please don't open these files in such a program, and don't alter them in any way. You can open them in a text editor to see their contents.*) You will need to implement the use of command line arguments to acquire the name of the tile to open each time your program is executed.

Ex. **python CPSC217S21A4-Hudson.py DataSet1.txt**

Each line in the file will be of the form **<SickPerson1>,<Contact1>,<Contact2>,...,<Contactn>**

<**SickPerson1>,<ContactX>** are placeholders for specific people.

Each line will always begin with exactly one person, followed by one or more other people. For example, lines in the file could be **Jonathan, Alice, Bob** or **Carol, Alice**

Note the names may include a mixture of upper and lowercase letters and may also include spaces. The end of a name is indicated by the comma separators. In order to make the files easier to work with you can assume that there will not be any spaces immediately before or after any of the commas (only within the start and end of a name).

Part 1: Who did each sick person have a record of contact with?

Your first task is to list everybody that each sick person had contact with using nice formatting. For example, if your file contains the line:

Jonathan, Alice, Bob

then you should output a single line for Jonathan that reads

Jonathan had contact with Alice and Bob

There are two parts to this printing. First is the name of the person for which we were printing out their contact record. Then in between are the words 'had contact with'. Finally, what follows is a formatted list of the people contacted by the first person.

*Notice that commas appear after all items except the last and second last items, and that the word "and" appears between the last and second last items. You will be graded on correctly following this layout, but I have provided a module **formatList.py** that includes a function that will do the formatting for you – all you need to do is import the function and call it. You can import this python file like you have **SimpleGraphics.py** previously and call the function **formatList(yourList)** to produce a string version of your list formatted as required above.*

In order to complete this and subsequent tasks you must load all of the data from the input file into a dictionary that describes the contact relationship. The keys in the dictionary will be the names of the first sick person. The values in the dictionary will be lists, where each element in the list is the name of a person that the sick person had contact with.

The output for Part 1 for DataSet1.txt should be:

```
Contact Records:
  Bob had contact with Carol, Leanne, Mark, Paul and Will
  Carol had contact with Leanne and Mark
  Farley had contact with Paul
  Leanne had contact with Sarai
  Larry had contact with Carol, Leanne, Mark and Will
  Mark had contact with Philip and Zach
  Paul had contact with Zach
  Will had contact with Leanne and Mark
  Zach had contact with Philip
```

Note: I have displayed my output in sorted order by using Python's sorted function. However, the order is not required. You'll receive full credit as long as all of the contacts are identified correctly.

Please remember to print the header **Contact Records** before your lines.

Part 2: Identify the Possible Patient Zero(s)

If we consider the input file to only consist of contact tracing records where we know everyone tested positive before constructing their contact record list, then we can track back the path of likely infection to what we will call **patient zero(s).** One way to think of **patient zero(s)** is that they are people that are sick who do not appear in anyone else's contact list. After displaying the listing described in Part 1 your program should continue and list the **patient zero(s)** for the input data with the appropriate heading.

The output for Part 2 for DataSet1.txt should be:

```
Patient Zero(s): Bob, Farley and Larry
```

Part 3: Identify the Potential Zombies

We will define a **potential zombie** to be any person that might be infected (because they have been in contact with a sick individual) but who has not yet been conclusively determined to be sick. Remember that a sick person occurs as a first entry in each contact record in the data file so a **potential zombie** will occur in the list of a sick person, but not occur as a sick person themself. The output from your program should continue by displaying all of the **potential zombies** with an appropriate heading. There should be no duplicates in this list.

The output for Part 3 for DataSet1.txt should be:

```
Potential Zombies: Philip and Sarai
```

Part 4: Identify People That Are Neither Patient Zero(s) nor Potential Zombies

The people that are neither patient zero(s) nor potential zombies are all of the individuals that occur in the data file but were not listed in part 2 or part 3. You can use the information returned by the functions created for those two parts to construct a list of names that only includes individuals who were not **potential zombies** and were not **patient zero(s)**. Then your function should return this list.

The output for Part 4 for DataSet1.txt should be:

```
Neither Patient Zero or Potential Zombie: Carol, Leanne, Mark,
Paul, Will and Zach
```

Part 5: Identify the Most Viral People

We will define the **most viral people** as the people who likely infected the greatest number of other people in the data set because they had the longest list of contacts. Identify and display all of the most viral people under an appropriate heading.

The output for Part 5 for DataSet1.txt should be:

```
Most Viral People: Bob
```

While DataSet1.txt has only one most viral person other data sets may have several most viral individuals, all of whom infected the same amount of people. When that situation occurs, your program should display all of the most viral people.

Part 6: The Most Contacted Person

We will define the most contacted person as the member of the data set that has possibly been infected by (been in contact with) the most sick members of the data set. Identify and display all of the **most contacted people** under an appropriate heading. Remember, there should be no duplicates in this list.

The output for Part 6 for DataSet1.txt should be:

```
Most Contacted: Leanne and Mark
```

Part 7: Determine the Maximum Distance from a Potential Zombie

We will define the maximum distance from a potential zombie as the longest contact tracing path downwards in the data set from a sick person to a potential zombie. Using this definition, all potential zombies (people that are not yet confirmed to be sick) have maximum distance 0. Any person that only has contact with potential zombies has maximum distance 1. All other people in the data set have a maximum distance which is one more than the maximum distance of the person they've had contact with who has the largest maximum distance value.

The description of maximum distance above is recursive – it defines the maximum distance of one person in terms of the maximum distance of another person. As such, you may find yourself wanting to write a recursive function to assist you with determining the maximum distance of each person in the contact tracing data. However, we will only cover recursion briefly at the end of this course, and we will not discuss it until the final week of classes. As a result, if you choose to develop a recursive solution you should expect to do some independent study on recursion (Chapter 12 in the third or fourth edition of Starting Out with Python).

If you don't want to do some independent study on recursion then you can determine the maximum distances of all the people in the contact tracing data using the following algorithm:

```
set the max distances of all people, including potential zombies, to 0
set changed to true
while something has changed
      set changed to false
      for each person, p1, in the dataset
            for each person, p2, that p1 had contact with
                  if the max dist of p1 <= max dist of p2
                        set the max dist of p1 to the max dist of p2 + 1
                        set changed to true
```

The output for Part 7 for DataSet1.txt should be:

```
Heights:
  Bob: 4
  Larry: 4
  Carol: 3
  Farley: 3
  Will: 3
  Mark: 2
  Paul: 2
  Leanne: 1
  Zach: 1
  Philip: 0
  Sarai: 0
```

Requirements

- **Your program must read the data file name as a command line argument.** The dataset file will be the first and only command line argument, appearing in sys.argv[1]. **If no command line argument is provided, then your program should read the name of the file from the user.** If more than 1 command line argument is provided, then your program should display an appropriate error message and quit.
- Your program must perform basic error checking, such as ensuring that the file specified by the user exists. If an error is encountered while opening a file, then your program should display an appropriate error message and exit.
- **You must store the sick person contact relationships in a dictionary where the keys are the sick person names and the values are lists of people contacted by the sick person.**
- You must only read each file once. It is not acceptable to submit a solution that requires the file to be read several times to complete the analysis.
- Close every file that you open.
- Your program must make appropriate use of functions. Specifically, you should write one function for each part of the assignment for parts 2 through 7. (You can also write one or more functions for part 1 if you want to). **Your solution will not receive full credit unless your code is divided into appropriate functions that make appropriate use of parameters and return values. A significant deduction will be made if your program is all in one function, even if it generates beautiful results.**
- **The only lines of code in your program that should be outside of a function definition are constants (if any), import statements (if any) and the call to the main function (which is normally the last line in the file).**
- Do not define one function inside another function.
- Your program must not use global variables (except for constant values that are never changed, and in this assignment, you may not even find that you want any global

constants). If you load some data inside a function, then you must return it as a result so that it can be used in subsequent functions.

- You may assume that the data in the files you are working with is correct. Once you open the file successfully you don't need to worry about problems such as reading a number where a word is expected, a missing comma, a blank line, etc.
- Your program must use good programming style. This includes things like appropriate variable names, good comments, minimizing the use of magic numbers, etc. Your program should begin with a comment that includes your name, student number, and a brief description of the program. Each function should begin with a comment that describes its purpose, parameters (if any) and return values (if any).
- Break and continue are generally considered bad form. As a result, you are NOT allowed to use them when creating your solution to this assignment. In general, their use can be avoided by using a combination of if statements and writing better conditions on your while loops.

Dealing with Command Line Parameters inside IDLE (not relevant to PyCharm):

If you are working in IDLE (Python installed IDE) instead of directly from the command prompt, you might find yourself wondering how to provide command line parameters when running your program. Unfortunately, IDLE doesn't provide a good option for specifying command line parameters. Instead, the best strategy is probably to **TEMPORARILY** 'fake it' by adding the following line right after import sys:

sys.argv = ["CPSC217S21A4-Hudson.py", "DataSet1.txt"]

This will overwrite whatever is in sys.argv with a list that has the name of your .py file as the first element and the name of the file that you want to process as your second element**. You'll need to take this out before you submit the assignment because your TA will run your submission from the command prompt**, but you should be able to test all of the cases required for the assignment by changing the values in this list:

• You can test different files (including files that don't exist) by changing the second element in the list.

• You can test the case where the user provides too many command line parameters by adding another element to the list.

• You can test the case where the user doesn't provide a command line parameter by removing the second element from the list.

Hints:

Develop an algorithm for Part 1 before trying to write the code. The algorithm should have "For each line in the file" as an outer loop, followed by a description of the steps that you are going to follow to process the line and get the **sick person** and **contacts** into the dictionary correctly.

Then you'll have a separate loop which will displays the relationships that you stored in the dictionary.

Begin with Part 1 – all of the other parts rely on it. Don't move on to the other parts until you have correct lists in Part 1.

Parts 2 through 7 can be performed in any order. Part 5 is probably the easiest to complete. Part 7 is probably the most difficult.

Having trouble formatting the output in Part 1? There's a module on the course website that you can import to do the formatting for you.

Submit the following using the Assignment 4 Dropbox in D2L:

1. CPSC217F20A4-Name.py

Grading:

This assignment will be graded on a combination of functionality and style. A base grade will be determined from the general level of functionality of the program (Does it output the **contacts** for each **sick person**? Is it formatted correctly? Does it identify the **patient zeros** and **potential zombies**? Does it identify the **most viral people** and the **most contacted people**?). The base grade will be recorded as a mark out of 12.

Style will be graded on a subtractive scale from 0 to -3. For example, an assignment which receives a base grade of 12 (A), but has several stylistic problems resulting in a -2 adjustment will receive an overall grade of 10 (B+). Fractional grades will be rounded to the closest integer.

Mark	Letter
12	A
11	A-
10	B+
9	B
8	B-
7	C+
6	C
5	C-
4	D+
3	D
0-2	F

Bonus

Identify the Spreader Zombie, Regular Zombie, and Zombie Predator:

A spreader zombie is sick person that only has had contact with potential zombies. Display the spreader zombies under an appropriate heading. If there aren't any spreader zombies then you should display the heading, followed by "(None)".

A regular zombie is a sick person that has had contact with both potential zombies and people who are already sick. Display the regular zombies under an appropriate heading. If there aren't any regular zombies then you should display the heading, followed by "(None)".

A zombie predator is a person that only has contact with people who are sick. Display the zombie predators under an appropriate heading. If there aren't any zombie predators then you should display the heading, followed by "(None)".

The output for spreader, regular, and predator zombies for DataSet1.txt are shown below:

```
For an A+:
  Spreader Zombies: Leanne and Zach
  Regular Zombies: Mark
  Zombie Predators: Bob, Carol, Farley, Larry, Paul and Will
```