Academic Dishonesty Disclaimer

All of the work you submit must be done by you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. The department uses software that compares programs for evidence of similar code. Please read the Rules and Regulations from the U of T Governing Council (especially the Code of Behaviour on Academic Matters).

Please don't copy. We want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

Never look at another assignment solution, whether it is on paper or on the computer screen. Never show another student your assignment solution. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in CS involve students who have never met, and who just happened to find the same solution online. If you find a solution, someone else will too. The easiest way to avoid plagiarism is to only discuss a piece of work on the course discussion board, with the CSC108H TAs in the open labs, or the CSC108H instructors in office hours.

Assignment 3

Please read this document very carefully. Follow instructions exactly. If you have any questions please post them to the course Piazza page.

This assignment is due December 5th, by 10:00pm

Imagine you work for a social media company named 'Chirper.' On this social media platform users have profiles and publish relatively short text messages for other users to read, like, and/or dislike; these messages are referred to as *chirps*. You are tasked with implementing functions which parse and manipulate the data of this social network platform. You are given a file, `assignment3.py`, with six incomplete functions. For this assignment, **you are required to complete these functions**. A description regarding the intended behaviour of these functions follows.

Note 1: One potential difficulty of this assignment is to realize how the data is stored and what it represents. Pay close attention to the input and returns types of each function.

Note 2: Due to the way data is represented, having exhaustive doc-string examples is challenging. It is your responsibility to ensure your function meets its specifications; not just that it passes the test case(s) covered by the docstring example(s).

To give you a better idea to how the social platform works, here are some summary points and definitions.

1. A *chirp* is string of length 280 or less. A chirp may also have *tags* associated with it which users can add. These tags are added to the end of each message with %'s in front of them.

   For example, a chirp may be: `'Nothing on the midterm was taught in lecture!  %MAT102 %Unfair  %ShouldHaveGoneToWaterloo'`

2. Each Chirper user has a set of profiles they follow, and a set of profiles which follow them. If User 1 follows User 2, it does not necessarily mean User 2 follows User 1.

3. A `profiles_dictionary` is of the type `Dict[int, Tuple[str, List[int], List[int]]]`; `profiles_dictionary` is used as a shorthand for this type in the function descriptions later in this document. The keys of a `profiles_dictionary` are unique integers representing user id numbers. The `profiles_dictionary` values contain the profile information corresponding

to the specific user id number given as the key. The values are Lists, where the *first element* of the list is the corresponding *user name*, the *second element* is a list of user ids representing the user's *followers* (the user id given as the key), and the *third element* is a list of user ids that the user in question is *following*.

4. A `chirp_dictionary` is of the type `Dict[int, Tuple[int, str, List[str], List[int], List[int]]]`; `chirp_dictionary` is used as a shorthand for this type in the function descriptions later in this document. The keys of a `chirp_dictionary` are unique integers representing chirp id numbers. The `chirp_dictionary` values contain the chirp information corresponding to the specific chirp id number given as the key. The values are Lists. The *first element* of the list is the *user id chirp's author*. The *second element* is *the chirp* itself. The *third element* is a list of *tags* associated with the chirp. The *forth element* is a list of user ids which *liked* the chirp. And the *fifth element* is a list of user ids which *disliked* the chirp.

Functions

You are required to implement all of the following functions. Pay attention to parameters of each function. For example, *if it is said an input will be an elevation map, you can trust your function will never be tested on input which isn't an elevation map.* For further examples of how these functions are intended to operate, view the docstrings of the starter code for this assignment.

1. `create_profile_dictionary(str) -> profiles_dictionary`
   The input parameter is the name of a text file containing all of Chirper's profile information. You may assume the file is in the same directory as `assignment3.py`. The file is a series of profile data, where data for each profile has the following format:

   ```
   USERID
   USERNAME
   FOLLOWER1, FOLLOWER2, ..., FOLLOWERn
   FOLLOWED1, FOLLOWED2, ..., FOLLOWEDm
   ```

   where `USERID`, `FOLLOWERi`, and `FOLLOWEDj` are all unique numbers. The *third line* represents the users which `USERNAME` *follows*, and the *forth line* represents the users which *follow* `USERNAME`. In the text file there is always a blank-line between different profiles. If the profile in question does not follow any users, or if no users follow them, the corresponding lines will be blank. See `profiles.txt` as an example.

   Based off this data, the function constructs a `profiles_dictionary` where the orders of the ids representing followers and users follows, are the same orders as which they appear in the text file. See the doctstring and `profiles.txt` for further clarification. Assume that the given text file is perfect and that there are no formatting errors.

2. `create_chirp_dictionary(str) -> chirp_dictionary`
   The input parameter is the name of a text file containing all of Chirper's message information. You may assume the file is in the same directory as the as `assignment3.py`. The file is a series of message data, where data for each message has the following format:

   ```
   CHIRPID
   USERID
   MESSAGE
   TAG1, TAG2, ..., TAGk
   LIKED1, LIKED2, ..., LIKEDn
   DISLIKED1, DISLIKED2, ..., DISLIKEDm
   ```

where `CHIRPID`, `USERID`, `LIKEDi`, and `DISLIKEDj` are all unique numbers. `USERID` is the id of the user which made the chirp. The *third line* is the chirp itself. The *forth line* represents a list of tags associated with said chirp. The *fifth line* is a sequence of user ids which *liked* the chirp. The sixth line is a sequence of user ids which *disliked* the chirp. If a chirp has no tags, likes, or dislikes, the corresponding lines will be blank. In the text file there is always a blank-line between different chirps. See `chirps.txt` as an example.

Based off this data, the function constructs a `profiles_dictionary` where the orders of the tags and ids representing likes and dislikes, are in the same orders in which they appear in the text file. See the docstring and `chirps.txt` for further clarification. Pay close attention to how the data is stored if there are no tags associated with a chirp. Assume that the given text file is perfect and that there are no formatting errors.

3. `get_top_chirps(profile_dictionary, chirp_dictionary, int) -> List[str]`
   The third parameter is a user id which is in the given `profile_dictionary`. Let the user with this user id be denoted by u. The function uses the data from the first two parameters to find the chirp with the most likes for each user followed by u. The function creates a list of these chirps and returns it. You may assume:

   (a) For a given user, there does not exist any two chirps with the same number of likes.

   (b) The order of the chirps in the returned list is arbitrary. As long as all the correct chirps are present and no incorrect chirps are present, the function is correct.

   See the docstring for examples.

4. `create_tag_dictionary(chirp_dictionary) -> Dict[str, Dict[int, List[str]]]`
   Takes in a `chirp_dictionary` and creates a new dictionary. In the new dictionary, tags are keys, the values are dictionaries where the keys in these dictionaries are user ids, and the values of these dictionaries are chirps made by the corresponding user, which also had the corresponding tag. See the docstring for examples.

5. `get_tagged_chirps(chirp_dictionary, str) -> List[str]`
   Takes in a `chirp_dictionary` and a tag, and returns a list which contains a list of all the chirps which had said tag on them. The order of the returned list is arbitary, as long as all the correct chirps are present and no incorrect chirps are present, the function is correct. See the docstring for examples.

Submitting and Grading

This assignment will be submitted electronically via MarkUs. Please find the MarkUs link on the course website. Your submitted file must have the name `assignment3.py`, or you will receive 0.

This assignment is worth 10% of your final grade. Grading is done completely automatically. That is, a program calls your function, passes it certain arguments, and checks to see if it returns the expected output. Each function is worth 20% of your assignment grade. For any one function, if you pass n of the m tests we run on that function, your grade for that function will be n/m.

Shortly after the deadline, you will receive your grade. If you are not content with this grade, you may resubmit your assignment up to 48 hours after the original deadline with a 20% penalty. If you choose to resubmit, your final grade on the assignment will be the higher of the two grades (the original submission, and the re-submission with a 20% penalty). Good luck!