Assignment 3

Due Monday by 4pm **Points** 8

Nov 21: revised **a3_checker.py** (updated formatting of tweet data to match the handout)

Nov 19: revised **tweets_big.txt** (added one missing end of tweet sentinel)

CSC108: Assignment 3

Tweet Analysis

Deadline: December 3, 2018 by 4:00pm

There is no resubmission for Assignment 3. Before submitting your assignment, run the checker and fix any errors (e.g, `SyntaxError`s) that prevent your code from running.

Introduction

In this assignment, you will analyze a number of tweets to see how organizations like UofT use Twitter. This assignment is designed to give you practice with dictionaries, files and and testing.

Here are definitions of terms that are used in the assignment:

Definitions

- **tweet**: A message posted on Twitter. The message text is between 1 and 280 characters long (inclusive).
- **hashtag**: A word or phrase in a tweet that begins with the hash symbol. Twitter uses the number sign (`#`) as the hash symbol. `#UofT` and `#csc108` are two examples of hashtags on Twitter. Hashtags are used to label important words or terms in a tweet. For our purposes, a hashtag begins with the hash symbol, and contains all alphanumeric characters up to (but not including) the first non-alphanumeric character (such as space, punctuation, etc.) or the end of the tweet. A hashtag either begins a tweet or is preceded by whitespace. A hashtag must contain at least one alpha-numeric character.
- **mention**: A word or phrase in a tweet that begins with the mention symbol. Twitter uses the at-sign (`@`) as the mention symbol. Mentions are used to direct a message at or about a particular Twitter user, so the word or phrase should be a Twitter username (but for the purposes of this assignment, we won't check if the username is valid — we'll just assume it). For our purposes, the definition of a mention is very similar to that of a hashtag. A mention begins with the mention symbol, and contains all alphanumeric characters up to (but not including) the first non-alphanumeric character (such as space, punctuation, etc.) or the end of the tweet. A mention either begins a tweet or is preceded by whitespace. A mention must contain at least one alpha-numeric character.
- **URL**: An address to a resource (like a webpage) on the Internet. For example, `http://www.twitter.com` and

How to not be interviewed for an academic offence

"The truth is that on every campus, a large proportion of the reported cases of academic dishonesty come from introductory computer science courses, and the reason is totally obvious: we use automated tools to detect plagiarism"
Professor Ed Lazowska, chair of computer science and engineering at the University of Washington, **Why Computer Science Students Cheat (https://www.pcworld.com/article/194486/Computer_Science_Students_**
.

We are very good at detecting cheating. We hate finding it, though, so we have written this short guide on how to avoid being interviewed by us for an academic offence.

A typical penalty for a first offence is a **zero on the assignment**. The case will also be entered into the **UofT academic offence database**. (If you get caught a second time ever as an undergrad, the penalties are much, much more severe.)

Our tips:

- Remember that figuring out what code needs to be written is the most important part of the assignment. Typing out code based on steps that someone else gave you is considered cheating, as you are submitting someone else's ideas as your own.
- Don't search the web for solutions. We've already done that and will be comparing what we found with what you submit. It doesn't help to change variable names and move functions around. Our software will still find you.
- Don't ask your friend for their solution, even if you just want it to "see how to solve it". Don't show your friends your solution either.
- Don't use a solution from a previous year. We have them all, and will be comparing your submissions to them.

`https://bit.ly/2AJZMGR` are URLs. For our purposes, a URL either begins a tweet or is preceded by whitespace, starts with the four characters `http`, and contains all characters up to (but not including) whitespace or the end of a tweet.

For a complete list of Twitter terms, check out the **Twitter glossary (https://help.twitter.com/en/glossary)**.

- Don't get solutions, or partial solutions, from a tutor. They frequently provide the same code or ideas to more than one person. We know because we frequently catch people who do it.
- Only ask for detailed help on your code from official CSC108 course staff, including the Help Centre.
- If you can't figure out how to write a function, it's often because you haven't fully understood an underlying concept: aliasing? mutation? something else? Review the materials on that topic, experiment in the Python shell, and ask us for help!
- Remember that it is better to submit an assignment with a few missing functions than to cheat!

Files to Download

Please download the Assignment 3 Files (**a3.zip**) and extract the zip archive. A description of each of the files that we have provided is given in the paragraphs below:

Starter code:

- Main program: `tweets.py`
  The `tweets.py` file contains some constants, import statements, a main block, helper functions, and the docstring for the first required function. You are to modify this file by adding function definitions.
- Test Files: `test_extract_mentions.py` and `test_common_words.py`

  We are providing the beginning of two unit test files, `test_extract_mentions.py` and `test_common_words.py`. You are to modify these files to contain your unit tests for the `extract_mentions(...)` and `common_words(...)` functions, respectively, that are described below.

Checker: `a3_checker.py`

We have provided a checker program (`a3_checker.py`) that tests two things:

- whether your functions have the correct parameter and return types, and
- whether your code follows the Python and CSC108 **Python Style Guidelines**.

The checker program does not test the correctness of your functions, so you must do that yourself.

Data Files

We're providing two files:

- `tweets_small.txt`: a text file containing a small set of tweets made by two Twitter users, and
- `tweets_big.txt`: a text file containing a larger data set.

You **should not** use the larger data set for early testing: how will you know what the right result is? With the smaller data set, you can calculate what the result should be.

Both data files follow a particular format. Open the small data file with a text editor or Wing101 and compare the file with the description of the format described below.

The files contain a sequence of Twitter-user histories. Each Twitter-user history has the following high-level format:

```
USERNAME:
TWEET1
TWEET2
TWEET3
...
```

The end of a particular Twitter-user's history has been reached when a new Twitter-user's history begins or the end of the file has been found. Each Twitter-user's history begins with a single-word line that ends with a ":". Each USERNAME is an alphanumeric string. Each TWEET in a history contains many pieces of information, split across multiple lines. The first line contains interesting properties of the tweet. The lines that follow contain the text of the tweet message. The last line contains the string: "<<<EOT". Each TWEET in a history has the following format:

```
DATE,LOCATION,SOURCE,FAVOURITE_COUNT,RETWEET_COUNT
TEXT
```

```
...
<<<EOT
```

The tweet properties in the first line are comma separated. The first value is the tweet's DATE in the form YYYYMMDDHHmmSS (where Y is for the year, M is month, D is day, etc. This format allows you to use comparison operators to compare two tweets to find out which date is earlier.) The LOCATION and SOURCE describe the location where the tweet was made, and the app or device used to create the tweet. The FAVOURITE_COUNT and RETWEET_COUNT represent the number of times the tweet was 'favourited' or 'retweeted', respectively.

The TEXT of the tweet can span multiple lines, so the file contains a sentinel to indicate that the message has ended. The sentinel in our data files is a line that only contains "<<<EOT" (and a newline character). When a line only contains the sentinel, you know that all of the TEXT of the tweet has been read. The sentinel is not part of the tweet TEXT.

All tweets in a data file will be well-formed; you do not need to handle the case where a tweet or twitter-user history does not match the format described above.

Your Tasks

You are to write the required functions described below in the file `tweets.py`. You are to also write a suite of unit tests for two of the required functions.

This section contains a table with detailed descriptions of the functions that you must complete. These are the functions that we will be testing. You should follow the approach we've recently been using on large problems to break complex tasks into simpler tasks. Write helper functions to solve the simpler tasks. We will check your use of helper functions! Each helper function must have a clear purpose. Each required and helper function must have a complete docstring produced by following the Function Design Recipe. You should test all of your functions, both required and helpers, to make sure they work correctly. When possible, your functions must make use of the provided defined constants.

Functions to write for A3

Function name: (Parameter types) -> Return type	**Full Description (paraphrase to get a proper docstring description)**
`extract_mentions: (str) -> List[str]`	The parameter represents a tweet's text. This function should return a list containing all of the mentions in the text, in the order they appear in the text, converted to lowercase. Each mention in the returned list should have the initial mention symbol removed, and the list should contain every mention encountered — including repeats that occur when a user is mentioned more than once within the text. Hint: You may find the helper function `first_alnum_substring` useful.
`extract_hashtags: (str) -> List[str]`	The parameter represents a tweet's text. This function should return a list containing all of the unique hashtags in the text, in the order they appear in the text, converted to lowercase. Each hashtag in the returned list should have the initial hash symbol removed, and should be different from every other hashtag. Even if the text contains the same hashtag multiple times, it is to be included in the list only once. The order of the hashtags should match the order of the first occurrence of each tag in the tweet. Hashtags are case insensitive. This means that `#CompSci` and `#COMPSCI` are considered the same, and only one instance of `compsci` should appear in the result. Hint: You may find the helper function `first_alnum_substring` useful.
`count_words: (str, Dict[str, int]) -> None`	The first parameter represents a tweet's text, and the second represents a dictionary containing lowercase words as keys and integer counts as values. The function should update the counts of words in the dictionary. If a word is not the dictionary yet, it should be added. For the purposes of this function, words are delineated by whitespace: every string that occurs between two pieces of whitespace (or between a piece of whitespace and either the beginning or end of the tweet) could be a word. Numeric characters are treated the same way as alphabetic characters. Hashtags, mentions, and URLs are not considered words. The empty string is not considered a word. We will remove all non-alphanumeric characters from all words. For example, if we are analyzing the tweet "#UofT Nick Frosst: Google Brain re-searcher by day, singer

	@goodkidband by night!", we would increment the count for the word "by" by 2 and the counts for words "nick", "frosst", "google", "brain", "researcher", "day", "singer" and "night" by 1. Hint: You may find the helper function `clean_word` useful.
`common_words: (Dict[str, int], int) -> None`	The first parameter represents a dictionary of words to counts as described in `count_words` and the second represents a positive integer N. This function should update the given dictionary so that it only includes the most common words (i.e., the words that appear with the highest frequency). At most N words should be kept in the dictionary. If including all words with a particular word count would result in a dictionary with more than N words, then none of the words with that word count should be included. (That is, in the case of a tie for the Nth most common word, omit all of the words in the tie.)
`read_tweets: (TextIO) -> Dict[str, List[tuple]]`	The parameter refers to a file that is open for reading. The data in the file is in the Data Format described above. This function should read all of the data from the given file into a dictionary. The keys of the dictionary should be Twitter usernames converted to lowercase, and the items in the list associated with each username are tuples representing the tweets that user has sent. A tweet tuple should have the form (tweet text, date, source, favourite count, retweet count). The date, favourite count, and retweet count should be `int`s, and the other items in the tuple should be `str`s. Leading and trailing whitespace should be removed from the tweet text. Note: since this function takes a file open for reading as its argument, you do not need to provide docstring examples for it. This same rule applies as well to all helper functions with a type `TextIO` argument.
`most_popular: (Dict[str, List[tuple]], int, int) -> str`	The first parameter represents the dictionary produced by `read_tweets`. The second and third parameters are dates (expressed in the same `int` format as in the data file) with the second parameter less than or equal to the third parameter. This function should return the username of the Twitter user who was most popular on Twitter between the two dates (inclusive of the start and end dates). A user's popularity in a time period is the sum of the favourite counts and retweet counts for all tweets issued in that time period. In the case of a tie or no tweets in the date range, return the string `'tie'`.
`detect_author: (Dict[str, List[tuple]], str) -> str`	The first parameter represents a dictionary in the format produced by `read_tweets` and the second represents a tweet's text. This function should return the username (in lowercase) of the most likely author of that tweet, based on the hashtags they use. If all hashtags in the tweet are uniquely used by a single user, then return that user's username. Otherwise, return the string 'unknown'. Hint: For this function, we suggest creating a helper function that will find all hashtags for the users.

Required Testing (`unittest`)

Write (and submit) unittest testfiles for functions `extract_mentions(...)` and `common_words(...)`. These tests should be implemented in the appropriately named starter files.

We will evaluate the completeness of your unittest testfiles by running them against flawed implementations we have written, to see how many errors you catch. Avoid redundant tests. The goal is to catch all of our errors without extra, unnecessary tests.

Your unittest testfiles should stand alone. That is, they should require no additional files (like a tweets file). Instead, you should define appropriate test values (for example, a dictionary that might be generated from reading a tweets file) in the testfile and use them in your tests. You may assume that the folder that contains your unittest testfiles also contains a `tweets.py` file.

Using Your Functions

Once your functions are implemented, here are some ideas about how you might end up using them. You're not required to do any of these things, but this should give you a sense of how these functions might be used.

You could call `read_tweets` and then `most_popular` to find out whose tweets were most popular in a particular time frame. You could also write a little more code to be able to pass each tweet in the dictionary produced by `read_tweets` to `count_words`, and then to use `common_words` to learn which words were most common for a particular users.

What Not to Do

- Do **not** call `print`, `input`, or `open`, except within the `if __name__ == '__main__'` block.
- Do **not** use any `break` or `continue` statements.
- Do **not** modify or add to the import statements provided in the starter code.
- Do **not** add any code outside of a function definition or the `if __name__ == '__main__'` block.
- Do **not** use any global variables (other than constants).
- Do **not** mutate objects unless specified.
- Do **not** use Python language features for sorting that we haven't covered in this course, like the optional parameter `key` or the function `sorted`.

Marking

These are the aspects of your work that will be marked for Assignment 3:

- **Correctness (70%):** Your functions should perform as specified. Correctness, as measured by our tests, will count for the largest single portion of your marks. Once your assignment is submitted, we will run additional tests, not provided in the checker. Passing the checker **does not** mean that your code will earn full marks for correctness.
- **Testing (10%):** We will run the `unittests` that you submit on a series of flawed (incorrect) implementations we have written. Your testing mark will depend on how many of the flawed implementations your unittests catch, whether they successfully pass a working (correct) implementation, and whether your test files contain redundant (unnecessary) tests.
- **Coding style (20%):**
  - Make sure that you follow **Python Style Guidelines** that we have introduced and the Python coding conventions that we have been using throughout the semester. Although we don't provide an exhaustive list of style rules, the checker tests for style are complete, so if your code passes the checker, then it will earn full marks for coding style with two exceptions: docstrings and use of helper functions may be evaluated separately. For each occurrence of a **PyTA error** (http://www.cs.toronto.edu/~david/pyta/), a 1 mark (out of 20) deduction will be applied. For example, if a C0301 (line-too-long) error occurs 3 times, then 3 marks will be deducted.
  - Your program should be broken down into functions, both to avoid repetitive code and to make the program easier to read. If a function body is more than about 20 lines long, introduce helper functions to do some of the work -- even if they will only be called once.
  - All functions, including helper functions, should have complete docstrings including preconditions when you think they are necessary.
  - Also, your variable names and names of your helper functions should be meaningful. Your code should be as simple and clear as possible.

What to Hand In

The very last thing you do before submitting should be to run the checker program one last time.

Otherwise, you could make a small error in your final changes before submitting that causes your code to receive zero for correctness.

Submit `tweets.py`, `test_extract_mentions.py`, and `test_common_words.py` on **MarkUs** (https://markus108.teach.cs.toronto.edu/csc108-2018-09) by following the instructions on the syllabus. Remember that spelling of filenames, including case, counts: your file must be named exactly as above.