**CS5001**

**Spring 2019**

**Homework 6**

**Assigned:** March 14, 2019

**Deadline:** March 26, 2019 at 9:00am

Note that this homework is due on a Tuesday!

You have about a week and a half to work on it, then a little break before HW7 is assigned.

To submit your solution, compress all files together into one zip file. Login to handins.ccs.neu.edu with your CCIS account, click on the appropriate assignment, and upload that single zip file. You may submit multiple times right up until the deadline; we will grade only the most recent submission.

Your solution will be evaluated according to our CS5001 grading rubric. This is a programming-only assignment!

Filenames are not specified; use whatever you need to make it work. **Do not submit any .txt files.**

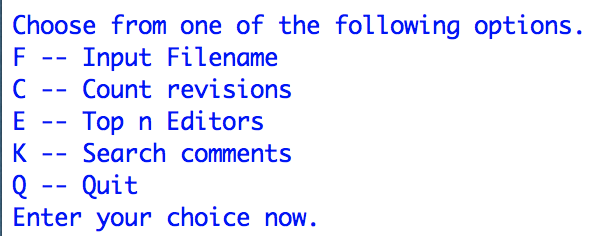
If you apply your late token, your new deadline is **March 30 at 9:00am**. Email laneys@northeastern.edu to notify us you’ll be cashing it in.

**The Assignment**

Every time a [Wikipedia](https://www.wikipedia.org/) article is revised, which is frequently (check out this fun snarky article on [Edits Per Day](https://en.wikipedia.org/wiki/Wikipedia:Edits_Per_Day)), the site generates some metadata -- data about the revision and the article that got revised. It stores things like the article number, the username of the person who did the editing, a timestamp, etc. All of this metadata is logged and stored forever on their servers.

For this assignment, we’ve gathered some wiki metadata, and you’ll create a program that allows the user to explore the information stored in a wiki metadata file ([see below for more on the file format](#kix.320y0dvqxttq)).

You’ll start by giving the user a menu of options:



If they enter an invalid option, repeatedly re-prompt them until they give you a good one. The only way to exit the program is if the user types Q to quit. The user should be able to enter upper or lowercase letters.

**F -- Input Filename**

When the user chooses this option, prompt them to tell you the name of the file where the wiki metadata lives. They’ll provide *just* a filename.

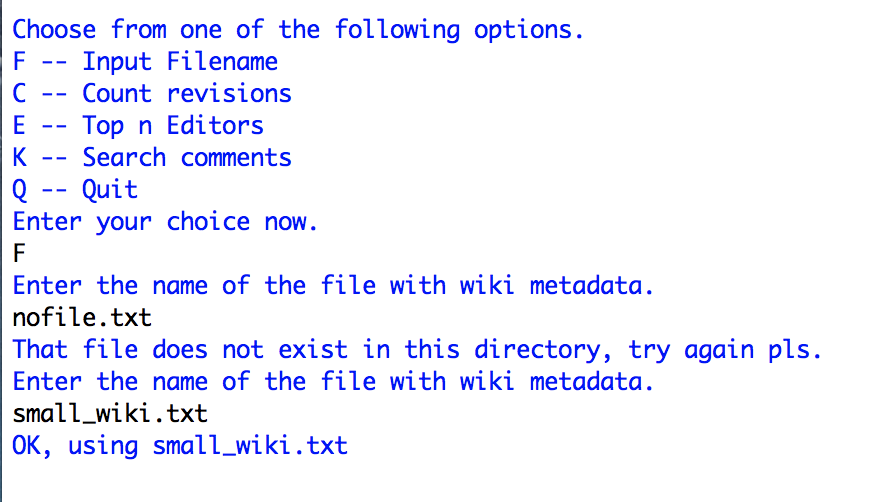
Check to make sure that the file exists in the same directory as the Python program you’re running. If the filename they’ve provided doesn’t exist in the same directory, repeatedly re-prompt until they enter a valid filename.

* Note that we didn’t cover this specifically in class, so you’ll need to do a bit of research. Try looking up *os.path*.
* Checking to see if a file exists is **not** the same as opening the file. They are separate steps. If the file exists, then we can try to open/read/close it.

From that point on, every time the user picks another option off the menu, we’ll be working with the file they specified (unless they change it by choosing F again).

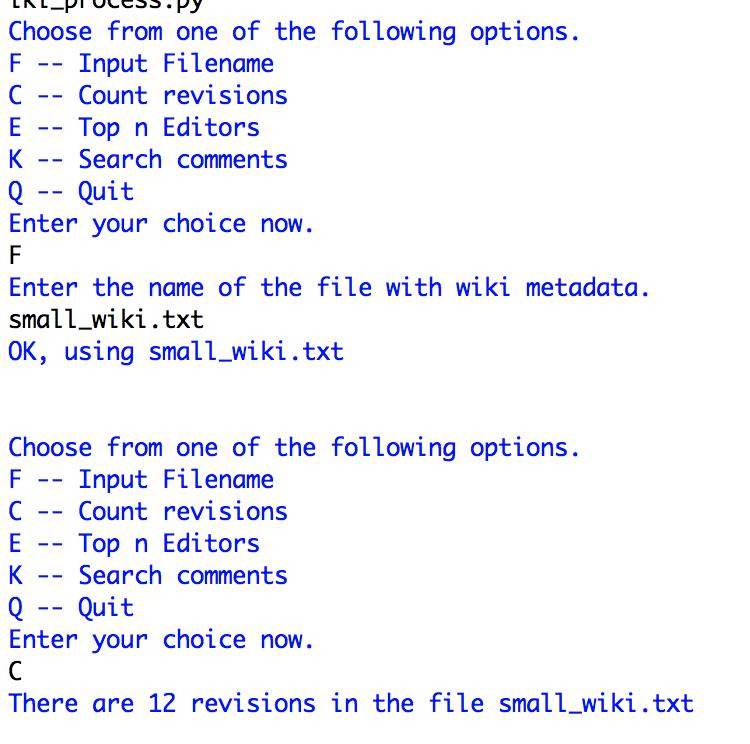
We expect the user to do this first. If they choose another option like Count Revisions or Top *n* Editors, then we won’t be able to process it because there is no file. If this eventuality happens, report the issue to the user but don’t quit, just return to the menu again.

For example, here’s what I would do as a user if I want to explore the file small\_wiki.txt:



**C -- Count Revisions**

When the user chooses this option, count the number of revisions logged in the file (see more on the file format, below). For example, here’s what I get when I choose this option and I’ve already chosen my file to be small\_wiki.txt:



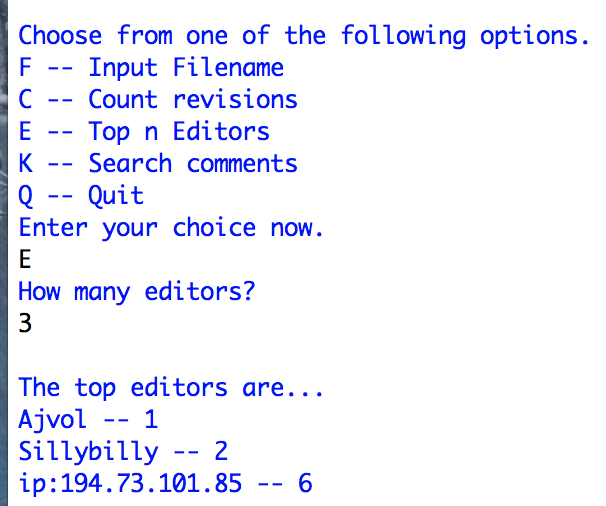
**E -- Top *n* Editors**

When the user chooses this option, we care about the number of revisions each editor has contributed. You’ll prompt the user for a number of editors, *n*, and then print the usernames of the *n* editors who have made the most revisions in the file.

Editors are identified by both username and ID number, but the username is more readable for us hoo-mans so we’ll use that. (See more on file format [below](#kix.320y0dvqxttq).)

A valid value for *n* is 1 to num-editors, inclusive. If the user enters a number less than 1 or if they enter a non-integer value, **don’t** re-prompt them but instead just use the value *n* = 1. If the user enters a number greater than the total number of editors, **don’t** re-prompt them but instead just use the value *n* = number of editors.

For example, here’s what I get when I choose this option and my file is small\_wiki.txt:



Why do I get this output? Because the file small\_wiki.txt has 12 total revisions, made by the following editors:

* **Taejo** made one revision
* **ip:66.112.232.43** made one revision
* **Ajvol** made one revision
* **Sillybilly** made two revisions
* **ip:194.109.243.248** made one revision
* **ip:194.73.101.85** made six revisions

So our top two editors are **ip:194.73.101.85** and **Sillybilly.**Then we have a four-way tie for the rest -- everyone else made one revision. So I just pick one at random, to round out the top three; it’s arbitrary which one I pick.

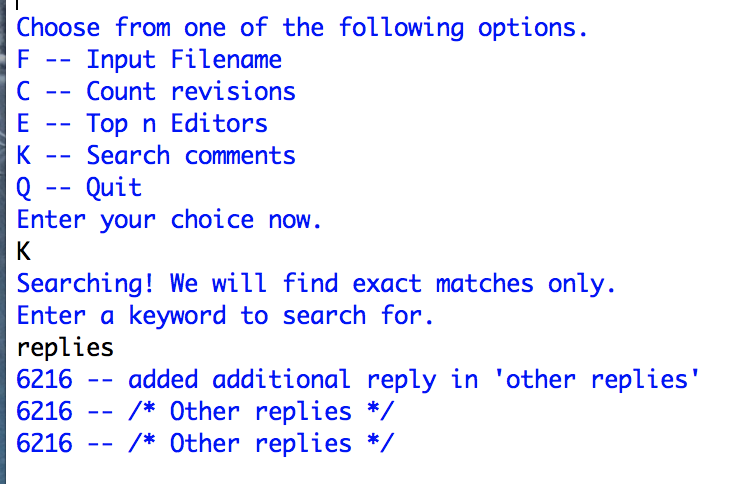
The order doesn’t matter. Above, my editors are in ascending order by number of contributions, but as long as you print them all, we don’t care about the order.

**K -- Search Comments**

*K* for keyword! If the user chooses this option, you’ll prompt them to enter a keyword and print out all the comments that contain it. We’re doing exact matches only; don’t worry about converting upper to lowercase, or punctuation. You may assume good input on this one -- that the user types a single word to search for.

Print both the article ID(s) and the full text of the comment(s) where the word was found. If the word was not found, report that no comments were found with that word.

For example, still using small\_wiki.txt as my file, here’s what would happen if the user searches for the word “replies”:



Because the word “replies” shows up in three revisions, but they’re all in the same article (with ID 6216). We print the article ID and the full comment for each instance we found.

**Q -- Quit**

End the program.

**Other Requirements**

* Anytime you process a file, you must use a **try/except** to catch any potential errors.
* Every time you open a file, you must also close it.
* As we noted above, you must use *os.path* to determine whether the file exists at all, and repeatedly re-prompt the user if it does not. (This is different than the try/except -- with *os.path* we check to see if a file exists without actually opening it or reading from it.)
* Allow the user to enter menu choices in upper or lowercase.
* The program should exit only when the user types Q to quit.

**Testing Requirements**

You must submit a test suite that uses only small\_wiki.txt as its file. (I.e., when we run your test program, assume we have small\_wiki.txt in that same directory but no other files.) Test any and all functions you write, in a test suite with its own main. When I run your test code, I should see on the terminal:

* The name of the function you’re testing
* The input you’re testing and its expected output
* Success or failure on that test
* At the end, total number of tests that succeeded/failed

As always, the goal here is to convince both yourself and us that you’ve made sure your functions are working ***before*** you plug them into a larger program. In “real life”, a software engineer tests their individual functions one at a time, and then hands them off to someone else to incorporate into another piece of code with full confidence that they work great -- that’s what we’re going for.

Here’s a recent example of test code from class/lab for reference: <http://www.ccs.neu.edu/home/laney/cs5001/spring19/code/test_recursive.py>

**The Input Files**

The Wiki metadata files follow a specific format. For each revision, you’ll find:

REVISION ArticleID RevisionID Title Timestamp EditorName EditorID

COMMENT Comment

If the editor doesn’t have a valid username or ID, you’ll see IP addresses for those fields. The editors’ usernames and ID numbers are both distinct. When reporting the top *n* editors, use the username because it’s friendlier to read.

(There are other items logged with each revision as well, but we can just ignore them because they’re not relevant to anything the user might do. REVISION and COMMENT are the only items we care about.)

We’ve provided three files of wiki metadata. The first two should be small enough that you can easily inspect them and make sure your code is working. The third is a little bigger, but still worth testing on. All three files have revisions for only *one* article -- 6216.

* [mini\_wiki.txt](http://www.ccs.neu.edu/home/laney/cs5001/spring19/code/hw6/mini_wiki.txt) (1 revision entry)
* [small\_wiki.txt](http://www.ccs.neu.edu/home/laney/cs5001/spring19/code/hw6/small_wiki.txt) (12 entries)
* [medium\_wiki.txt](http://www.ccs.neu.edu/home/laney/cs5001/spring19/code/hw6/medium_wiki.txt) (38 entries)

However! We’ll expect your program to run on *any* file as long as the format is OK. We’ll be grading your program in part by running it against a file you’ve never seen (and a pretty big one, a couple thousand entries probably). So with that in mind, you might want to run your program on some files beyond just the small-ish 3 given here.

There are some files with the correct format here: <http://snap.stanford.edu/data/wiki-meta.html>

**AMAZING points for this homework** (assuming everything else works and is lovely)

* Add another menu option -- could be anything, as long as it’s non-trivial -- and make it work.