CSCI 141 Final Project: An Online Forum

Due Thursday, December 15 2022 2200 EST (10:00 PM EST)

For this project, you will create a class called Forum_Page to represent an online discussion forum page, somewhat like a simplified version of Reddit. Users can post to the page, and can also vote on posts (messages) based on whether they like them or not. You will write a main program which creates an instance of this class and calls its methods to add posts, vote on posts, and perform other operations. A more detailed high-level description of how the online forum page works is provided below. Each Forum_Page object contains a Data Frame that holds a record of each post made, including the title, author, date, and the sum of its votes.

You have been provided with 3 files:

- Forum_Page.py: a skeleton file for the class definition with some methods completed and other methods for you to fill in
- words.txt: a text file containing words that will be used to auto-generate usernames (see further details in project description)
- Final_Project_Main.py: a skeleton file for your main program

How the Forum_Page works

The Forum_Page is a place where users can post questions/messages/topics for discussion. They provide a title and the main text of the post. We are going to require that titles be **unique** (they can't be repeated). When a user makes a post, they have the option to provide their username or to post anonymously. If they choose to post anonymously, a username will be randomly generated and listed with their post. Each post is marked with the date posted. The date can be automatically generated or provided by the user (see below).

Users are also able to indicate whether they like or dislike a post by voting. Votes always have a value of 1, so an upvote means adding one point to the vote total, a downvote means subtracting one point from the vote total.

Users can delete a post, but this is done in the style of Reddit: the author and main text of the post are removed, but the title and the date and time still remain. Posts that are deleted cannot be voted on.

To simplify object construction, we are not allowing users to edit posts after they are created or to reply to posts. The only actions they can take are to add posts, delete posts (as described above), and vote on posts.

Forum_Page Class Data

Your Forum_Page class will store the following data.

- **A name for the page.** When the object is created, this is a string specified by the user. It will not change after it is created in the constructor.
- **A list of words used to create anonymous usernames.** When the object is created, this is loaded from the file words.txt (provided for you) and is stored as a list, i.e. it is the text

from the file listed as individual words. It will not change after it is created in the constructor.

- **A Data Frame with information on posts.** This is a pandas data frame. The index of the Data Frame is the title of the post. The columns contain the date of the post, the author of the post, the full body of the post, and the vote total for the post. The data frame will start out empty when it is created in the constructor, and will gain entries as posts are added.

This is what the Data Frame looks like with a few entries:

Title	Date	Author	Post	Votes
A Post	2022-01-01	potato_lynx_2895	This post is about nothing	0
Override	2022-12-02	Mr. Potato	For students who need overrides to enroll in C...	0

The ... you are seeing represents further text that is too long to display.

Part 1: Forum_Page Class Methods

Your Forum_Page class will implement a variety of methods that allow modification of and access to the internal data. Some of these are already written for you, and you are not permitted to change them, so please read carefully.

Methods written for you

• ***Constructor***

This takes no arguments. It stores the name, and creates the list object that holds the words for the anonymous usernames, and the empty Data Frame that will hold the post information. **The constructor has already been written for you. DO NOT CHANGE IT!** You need to understand what it is doing and what the attributes it creates correspond to (see above).

• ***__process***

This is a private method. The argument, **filename**, is a string that represents a filename. For example, **filename** could be 'words.txt'. This method assumes that the file has one word per line. The method returns a list of words. This method is called by the constructor in the code given to you. It should not be called anywhere else in your class definition.

• ***__exists***

This is a private method. The argument, **title**, is a string that represents a post title. The method returns a boolean: True if the title is in the Data Frame, False if it is not. You will need to use this method in your other methods, and you MUST use this method where appropriate as opposed to repeating code from it.

• ***checker***

The public method **checker** is a tool for you to use while programming. It returns a copy of the internal data frame that you can use for debugging purposes. It should not be called by any of your other methods.

• ***get_name***

The public method **get_name** returns the name of the Forum Page that is stored in the object.

Methods you need to write

We expect your method implementations to reflect what you have learned about writing **concise, logical** code this semester. BE AWARE: Your project submission will be graded on style in terms of using pandas methods where appropriate and writing compact code as needed. There are also some suggestions about this in the method descriptions.

You will fill in your methods in the skeleton file **Forum_Post.py** provided. Your method implementations will be tested using a series of test cases, so be sure to test them thoroughly.

•***__generate_anon***

The private method **__generate_anon** generates anonymous usernames using the list of words created in the constructor. You only need to consider the words.txt file in this Project. The length of each word in the words.txt file ranges from 4 to 14.

There are three optional arguments: **maxlength** – the maximum length of the anonymous username, **min_num** – the minimum number of random digits, **max_num** – the maximum number of random digits. You can assume that the arguments if passed in will all be non-negative integers. Usernames are constructed as follows:

- Select two words randomly from the list of words, e.g., harmonious and turtle. Words DO NOT need to be unique, which means that you can select the same word twice.
- Select random digits between 0 and 9 inclusively. Digits need to be UNIQUE, which means that you cannot select the same digit. The number of digits is determined by the arguments.
- The anonymous username is the first word, an underscore, the second word, an underscore and then the digits, like this: harmonious_turtle_843
- Please note the anonymous username must have the first word, an underscore, the second word and an underscore. It can have 0 digits. For example, suppose the user asks for min_num = 0, max_num = 0, one of the valid usernames could be: potato_potato_
- You need to check if this username already exists (i.e., it has been used for a post previously). If it does, generate a new one. Keep generating a new random username until you get one that hasn't already been used. In this case, this method **returns** the username.
- However, if the user provides impossible specifications, for example, **min_num** exceeds the **maxlength**, or there is no room for the must-have (two words and two underscores), it should print an error message that reads 'Invalid Specifications' and return None.

You can assume that the words.txt file contains enough words that users need to generate anonymous usernames. If you are stuck on how to select things, a quick look at the documentation for the random module should answer your questions:

https://docs.python.org/3/library/random.html

• ***add_post***

The public method **add_post** allows users to add a post to the Forum_Page. **title**, a required argument, is a string that is the title for a post. Here is an example of what it might be: 'This is a POST'. **post**, another required argument, is a string that is the text of the post. Here is an example of what it might be: 'This is the actual text of the post, where I talk about my opinion on a thing.'.

It also takes five optional arguments. The optional argument **author** means the user can provide a string to be used as the author for the post. Here's an example of what it might be: 'a_turtle'. The optional argument **date** means the user can provide a string representing the date for the post in ISO 8601 format, YYYY-MM-DD. Here's an example of what it might be: '2022-01-01'.

The other three optional arguments are: **maxlength** – the maximum length of the anonymous username, **min_num** – the minimum number of random digits, **max_num** – the maximum number of random digits. You can assume that these arguments if passed in will all be non-negative integers.

Here is how the method works:

- If the user did not provide a value for **author**, generate a username using the private method __generate_anon. If the specifications are not valid, the username should not be added. Error checking for the specifications occurs in __generate_anon and MUST NOT be repeated in add_post.
- If the user did not provide a value for **date**, generate the date automatically using str(datetime.date.today()), which will also generate the date in ISO 8601 format, YYYY-MM-DD.
- New posts start out with 0 votes
- As long as the title hasn't already been used (hint: use one of the methods), add a row to the data frame for this new post, remember, the title should be the index; if the title has already been used, do nothing.

This method will modify the internal data frame and return nothing (yes, returning nothing is the same as returning None, but you should not have a line of code that says return None).

• ***delete_post***

The public method **delete_post** allows the user to delete a post according to the rules given in the introduction and repeated below. The required argument, **title**, is a string that is the title for the post to delete. Here is how the method works:

- Make sure that the post exists in the data frame (hint: use one of the methods); if it doesn't exist then do nothing
- Set the author and text (the actual post) to missing, the value for the title and number of votes should be left as they are. (Hint: what should we use to denote missing data in pandas?)

This method will modify the internal data frame and return nothing (yes, returning nothing is the same as returning None, but you should not have a line of code that says return None). Do not

over-complicate this. If you are writing more than 3-5 lines of code, you may be doing it wrong.

• ***vote_post***

The required argument **title** is a string that indicates the title of the post being voted on. The optional argument **up** determines whether the vote is an upvote (+1) or a downvote (-1). True corresponds to an upvote, False corresponds to a downvote. Here is how the method works:

- Make sure that the post 1) exists in the data frame (hint: use one of the methods), and 2) is available to be voted on – deleted posts cannot receive votes. If the post doesn't meet these criteria, do nothing.
- Add or subtract from the vote total accordingly. For example, if a post has 10 votes and up is True, the post would then have 11 votes.

This method will modify the internal data frame and return nothing (yes, returning nothing is the same as returning None, but you should not have a line of code that says return None). Do not over-complicate this. If you are writing more than 5-7 lines of code, you may be doing it wrong.

• ***find_author_by_keyword***

The required argument **keyword** is a string. This method will return a list of authors who have the **keyword** in their posts (You don't need to check whether the keyword is in the title or not). We don't care about the case of the **keyword**, i.e., this search is case-insensitive.

Note that this method does NOT modify the internal data frame. It just extracts information from it. Do not over-complicate this. You shouldn't write any loops in this method.

• ***get_post_date_info***

The public method **get_post_date_info** returns a string that contains the date information for the post with the title given by the required argument **title**. Be sure to verify that there is actually a post with that title in the data frame. Note that this method does NOT modify the internal data frame.

You have been given the first four lines of code for this method. **If you choose not to use these four variables, you will receive no credit for this method.**

For example, if we have the Data Frame like this:

Title	Date	Author	Post	Votes
A Post	2022-01-01	potato_potato_3	This post is about nothing	0
Final_Project	2022-02-03	lynx_potato_347	You are working on the CSCI 141 Final Project	0
Potato_Recipe	2022-03-03	lynx_potato_6	There are lots of ways to make potatoes	0

This method will return a string for **A Post**:
'A Post posted on Saturday, January 1, the 1**st** day of 2022'

This method will return a string for **Final_Project**:
'Final_Project posted on Thursday, February 3, the 34**th** day of 2022'

This method will return a string for **Potato_Recipe**:
'Potato_Recipe posted on Thursday, March 3, the 62**nd** day of 2022'

- In order to calculate the weekday of the date, you can use Zeller's congruence. Zeller's congruence is a formula that is used to calculate the weekday of a date. The formula is:

  h = (q + (13 * (m + 1)) // 5 + K + K // 4 + J // 4 - 2 * J) % 7

  where:

  - h is the weekday (0 = Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday),
  - q is the day of the month, m is the month (3 = March, 4 = April, 5 = May, ..., 14 = February). Note that January and February are counted as months 13 and 14 of the previous year. E.g. if it is 2 February 2010, the algorithm counts the date as the second day of the fourteenth month of 2009. (https://en.wikipedia.org/wiki/Zeller%27s_congruence).
  - K is the year of the century (year % 100), and
  - J is the zero-based century (year // 100).
- You need to consider the case for the leap year. A leap year is a year that is divisible by 4, but not by 100, unless it is also divisible by 400. For example, the years 2000 and 2400 are leap years, but 1900 and 2100 are not. In leap years, February has 29 days.
- Remember to add the correct ending/suffixes (st, nd, rd or th) to the day of the year. (https://www.grammarly.com/blog/how-to-write-ordinal-numbers-correctly/)

•***__str__method***

The **__str__** method returns a string that contains a header line in the format: Titles for name_of_Forum_Page, and then all the titles of **active** posts on the forum page, one per line. An active post is a post that has not been deleted. Note that this method does NOT modify the internal data frame.
For example, if we have a Forum_Page object called CSCI_141 and the Data Frame looks like this:

Title	Date	Author	Post	Votes
A Post	2022-12-03	potato_lynx_685	This post is about nothing	0
Final_Project	2022-12-03	lynx_potato_346	You are working on the CSCI 141 Final Project	3
Potato_Recipe	2022-12-03	NaN	NaN	0
Sea Turtle	2022-12-03	turtle_turtle_6	Sea Turtles migrate during different seasons	0
The turtle moves	2022-12-03	turtle_lynx_1	This is probably a Terry Pratchett reference	0

The string returned by this method should be:

Titles for CSCI_141:
A Post
Final_Project
Sea Turtle
The turtle moves

Part 2: Creating and using a Forum_Page Object

You will write a main program that makes use of your Forum_Page class. Your code for this part of the project goes in the file Final_Project_Main.py. You have been given a skeleton file with the correct import lines. Your code should be concise.

Your main program will carry out the following operations. All of these are achieved by creating a single Forum_Page. You will call methods on the object to meet these goals, adding in other programming structures as needed. If you are repeating code that appears in your methods in the main program, something has probably gone wrong.

1. Create a Forum_Page object called CSCI_141
2. You have been given a list of 5 titles and a list of 5 post texts. The first item in the title list goes with the first item in the post text list and so on. Use these lists to add 5 anonymous author posts to the Forum_Page with the following specifications: a maximum length of 15, a minimum of 2 digits and a maximum of 5 digits, using automatically generated date. Your code should be as efficient as possible, think about how to get the matched items from both lists most concisely.
3. Upvote the post titled Final_Project 5 times
4. Print out a list of all of the authors who have 'CSCI' in their posts
5. Delete the post titled Potato_Recipe
6. Print out a string containing the date information for the title 'Pandas'
7. Print out a string representation of the Forum_Page Object

Please be aware that this main program does not test all of your methods and doesn't thoroughly test most of them. Test your code thoroughly. Test beyond the main program and the Autograder. Try to break all of your methods.

SUBMISSION EXPECTATIONS

Because the name of the class, methods, and files **must be exactly as specified** in order for our testing to work, we have provided skeleton files for you to fill in. Be certain not to change any names, and do not modify the values and numbers of parameters for each specified method. We expect your code to be efficient and to make use of internal methods as needed as opposed to repeating code from these methods.

Forum_Page.py: The skeleton file attached to this project, but with your implementations added. The file must contain only the methods provided and you must not change their names. Do not include any additional code in this file.

Forum_Page_Main.py: The skeleton file attached to this project, but with your implementation added. The file must contain only the main program described in the instructions.

Final.pdf: A PDF document containing your reflections on the project. You must also cite any sources you use and mention everyone's name you work with. You may collaborate with up to three other students for projects. Please be aware that you must strictly follow the 'empty hand policy'; all code written must be your own. Programs copied in part or wholesale from the web or other sources or individuals will result in reporting of an Honor Code violation.

If your code contains structures NOT mentioned in class or readings, please include the

following in your write-up:
If it's a method:
(1) Apply it to one of the examples in lecture notes.
(2) Compute the Big-Oh running time.
(3) What does this method do? What should be its input and output?
(4) Why do you use this new method instead of the way we learned in class or readings or labs? What is the advantage?

If it's not a method, but a concept, e.g., recursive data structures,
(1) Apply it to one of the examples in lecture notes.
(2) Compute the Big-Oh running time.
(3) For recursive data structures, please specify base case(s) and recursive case(s).
(4) Why do you use it instead of the way we learned in class or readings or labs? What is the advantage?

You will receive credit **only** if your structure is significantly better than the one mentioned in class or readings. You will receive no credit if you get worse time and/or space complexity by introducing anything that is NOT mentioned in class or readings or if the program can be written in a more concise way using anything we learned.

POINT VALUES AND GRADING RUBRIC

Methods:

- add_post: 10 pts
- delete_post: 8 pts
- vote_post: 9 pts
- generate_anon: 10 pts
- find_author_by_keyword: 9 pts
- get_post_date_info: 12 pts
- __str__: 10 pts

Main program: 22 pts
Auto Grader: 8 pts
Writeup : 2 pts

RUBRIC FOR MANUAL GRADING

Grade Level	**Appropriateness**	**Style**	**Code Efficiency**
A	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; any references used cited in write-up	Follows Python style guidelines we have discussed to include appropriate naming of variables and spacing; Proper use and naming of constants	Only contains structures necessary to accomplish the task at hand; Uses the most concise implementation possible
B	Code uses structures and modules discussed in class; variable names meaningful; references cited in write-up	Largely follows Python style guidelines but may be a few minor deviations	Only contains structures necessary to accomplish the task at hand; Implementation is generally concise
C	Code uses structures and modules discussed in class with minor additions from outside sources (e.g. string formatting) appropriately cited; most variable names meaningful	Several deviations from Python style guidelines; constants not used properly	Contains a small number of unnecessary structures such as casts that are not needed; Implementation somewhat concise
D	Code uses some structures and modules taken from outside sources with citations; variable names and code structure hard to decipher	Only loosely follows Python style guidelines	Contains a large number of unnecessary structures such as unused variables and casts; Implementation somewhat concise but could be improved, code may be hard to follow
F	Code uses structures and modules largely from outside sources only; variable names hard to decipher; code written in other programming languages or Python 2.7	Code appears to have been written with no consideration for style	Contains many unnecessary structures; Code is difficult to follow and overly verbose