CSCI 140 Final Project: An Online Forum

Due Thursday, Nov 19 2020 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 (month, day and year), and the sum of its votes.

You have been provided with 4 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
- Forum_Viewer.py: a file to generate an optional visualization of your Forum_Page class

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, which is automatically generated (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.

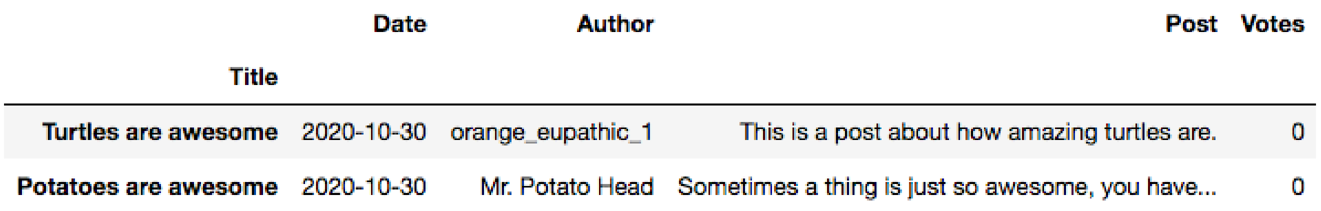
To simplify object construction, we are not allowing users to edit posts after they are created or to reply to posts. The only actions they can take are to add posts, delete posts (as described above), and vote on posts.

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 (do not freak out about the date, we give you the EXACT code to generate it below), 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:



	Date	Author	Post	Votes
Title				
Turtles are awesome	2020-10-30	orange_eupathic_1	This is a post about how amazing turtles are.	0
Potatoes are awesome	2020-10-30	Mr. Potato Head	Sometimes a thing is just so awesome, you have...	0

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

Some advice for working with a Pandas data frame:

- You can add rows to a data frame using the .loc attribute:

```
df.loc['new_row_name '] = [col1_info, col2_info, col3_info, etc.]
```

- If you want to look up things in the data frame, use the loc attribute:

`df.loc['row _name']` gives a Series containing the information associated with the row name.

`df.loc['row _name', 'column _name']` will give you the information from a single cell.

- You can access the index using: `df.index`
- Remember that every column in a data frame is a Series. Series have many useful methods. For example, you can get the maximum value by the calling the .max() method.
- The easiest way to find out how many rows are in a data frame is to use the built in function len. Use it the same way you would with any other object.
- The in operator works with columns in the data frame, for example you can ask if a value is in a column like this: `if value in df['column _name']`
- You can check if a value is np.nan by using: `pd.isnull(value)` – this assumes pandas is imported as pd. Check for a whole Series using the `.isnull()` or `.notnull()` methods.

It will save you a lot of time and effort to use this advice as opposed to searching the internet for ways to to do these things. That often leads down a complicated rabbit hole.

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*

The private method **__process** takes one argument, **filename**, 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*

The private method **__exists** takes one argument, **title**, 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. There are 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. It takes no arguments. Usernames are constructed as follows:

  - Select two **unique** words randomly from the list of words, e.g., harmonious and turtle, unique means that you cannot select the same word twice
  - Select two random digits between 0 and 9 inclusive, e.g. 8 and 4, digits DO NOT need to be unique (this means you could pick 8 and 8, or 4 and 4, etc.)
  - The anonymous username is the first word, an underscore, the second word, an underscore and then the digits, like this: harmonious_turtle_84
  - 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.
  - The method **returns** the username

There are 20 words and 10 digits, so creating the usernames this way allows for 38,000 possibilities! We shouldn't need more than that. If you are stuck on how to get unique words, 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 a user to add a post to the Forum_Page. It takes two required arguments: **title,** a string that is the title for a post, and **post,** a string that is the text of the post. It also takes the optional argument **author:** the user can provide a string to be used as the author for the post. Here's an example of what these arguments might be:

title: 'This is a POST'

post: 'This is the actual text of the post, where I talk about my opinion on a thing. For example, sometimes I'm very annoyed by fonts that have uneven kerning. They make writing up code with indentation that aligns very difficult'

author: 'a_turtle'

Here is how the method works:

  - If the user did not provide a value for **author** generate a username using one of your methods (hint, look at the previous page)
  - To automatically generate the date and time, use `str(datetime.date.today())`
  - 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). Do not over-complicate this. If you are writing more than 4 -5 lines of code, you may be doing it wrong.

- *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. It takes one required argument: **title,** 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

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 public method **vote_post** takes one required argument: **title**, 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.

- *top_voted*

The public method **top_voted** returns a data frame containing the most highly rated posts, i.e. posts which have the maximum number of votes. This may be a single post, or it may be multiple posts. For example, if 5 posts have 10 total votes and 10 is the maximum, **top_voted** would return a data frame containing all 5 posts. This method takes no arguments. Here is how the method works:

  - Find the maximum value of votes in the data frame (we mean the positive maximum)
  - Obtain all of the rows in the data frame with this number of votes and return them
  - Consider that sometimes the data frame may be empty and that might be a problem.

You may find consulting Project 4 helpful. Note that this method does NOT modify the internal data frame. It just returns a new data frame with selected information from the original. Do not over-complicate this. You can write this method in two lines (or one).

- *get_titles*

The public method **get_titles** returns a list that contains the titles of all of the posts in the data frame. This method takes no arguments. Make sure that your method returns a list.

Note that this method does NOT modify the internal data frame. It just extracts information from it. Do not over-complicate this. You can write this method in one line.

- *get_post_info*

The public method **get_post_info** returns a list that contains the information for the post with the title given by the required argument **title**. The list must include the date, author, text of the post, and vote total in that order (hint: that is to make it EASIER). Be sure to verify that there is actually a post with that title in the data frame. Make sure that your method returns a list.

Note that this method does NOT modify the internal data frame. It just extracts information from it. Do not over-complicate this. You can write this method in a few lines.

- *__str__ method*

The **`__str__`** method returns a string that tells how many active posts there are on the forum page. An active post is a post that has not been deleted. It also includes the name of the Forum_Page that is stored in the object. The string returned by this method should look something like:

100 active posts on name_of_Forum_Page

For example:

100 active posts on Turtle Central

The number 100 here is just an example. The number filled in will be the actual number of active posts. If you are stuck on how to count active posts, consider this: how do we identify posts that are active (or inactive)? Make a new data frame that contains just the active ones. Then count how many there are.

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.

- Create a Forum_Page object called CSCI_140
- 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; your code should be as efficient as possible, think about how to get the matched items from both lists most concisely.
- Upvote the post titled Final_Project 3 times
- Print out the top-voted post(s)
- Delete the post titled Potato_Recipe
- Print out a list of all of the post titles in the Forum_Page Object
- 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 testing script. Try to break all of your methods.

Optional: Visualizing the Forum_Page Interface

This section is completely optional. Whether you complete it or not has no bearing on your ability to complete the project and test your code. It is here for those of you who are interested and is kind of fun.

Be aware that some of you may have issues running the visualization and it will not run at all for you. Skip it if it doesn't run for you.

WARNING: If the visualization does not run for you, skip it - troubleshooting so that you can run the visualization can break your Anaconda installation completely.

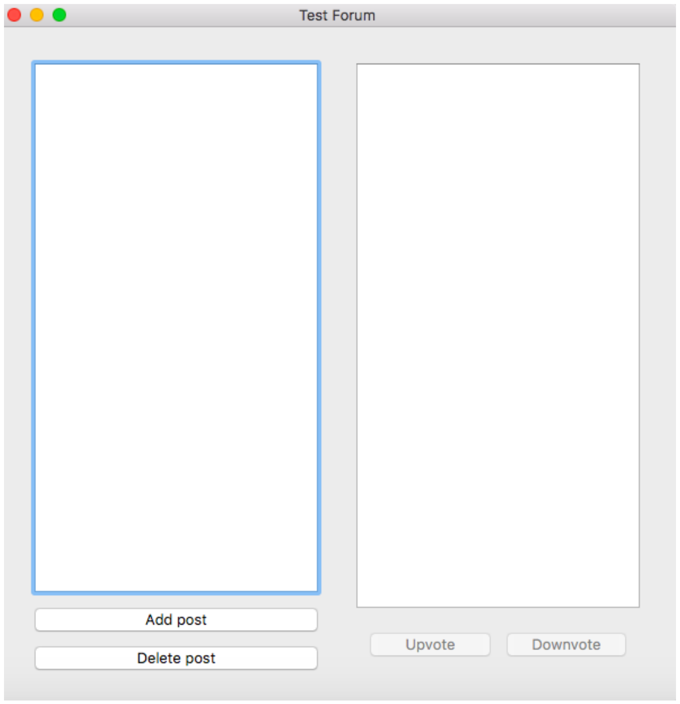
The file Forum_Viewer.py contains code to generate an interactive window that represents the forum page. In order to run this file, you must do the following:

- Save Forum_Viewer.py, your completed Forum_Page.py, and the word.txt file all in the same directory (folder).
- Navigate to the directory from the command line (Terminal/Powershell)
- In your terminal type:

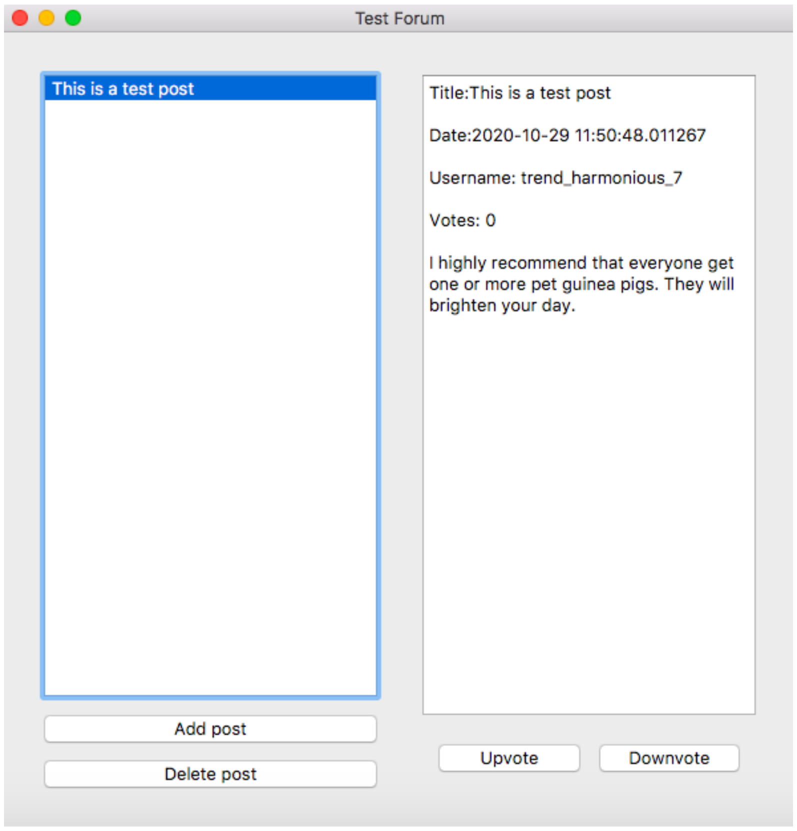
```
python Forum_Viewer.py
```

WARNING: If you try to run the Forum_Viewer.py file in a notebook, it will mostly likely crash your entire computer. You may lose unsaved work in Jupyter notebook. ONLY RUN FROM THE COMMAND LINE.

When you launch the Forum_Viewer, you should see an interface that looks like this:



If you click on Add post, it will bring up a dialog box where you can enter the Title, Author, and the Post. If you leave the author blank, it will automatically generate one...as long as your add_post method is written correctly. Once you add a post, it will appear in the window on the left. Click on the post name to see full information:



Title:This is a test post

Date:2020-10-29 11:50:48.011267

Username: trend_harmonious_7

Votes: 0

I highly recommend that everyone get one or more pet guinea pigs. They will brighten your day.

You can only upvote or downvote a post once per view. You'll see the buttons grey out after you do so. You'll have to select it again to vote again.

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 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_Post.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. You should fill in the parameter lists where indicated. Do not include any additional code in this file.

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

`Final.pdf`: A brief writeup describing any difficulties you encountered and how you overcame them. Also, cite any sources you may have used.

Please do not submit words.txt or Forum_Viewer with your project.

POINT VALUES AND GRADING RUBRIC

Methods:

-add_post: 4 pts

-delete_post: 3.5 pts

-vote_post: 4 pts

-generate_anon: 3 pts

-top_voted: 2 pts

-get_post_info: 1.5 pts

-get_titles: 1.5 pts

-__str__: 2.5 pts

Main program: 3 pts

Grade Level	**Execution**	**Correctness and Algorithms**	**Formatting**	**Style and References**
A	Code executes without errors, requires no manual intervention	Code correctly implements all algorithms, passes all test cases (with possible exception of extremely rare case), and meets all required functionality	Inputs and outputs formatted as per the specification both in terms of content and type, may be minor formatting issues (spacing, punctuation, etc.); correct file formats and names	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; any references used cited in write-up
B	Code executes without errors, may require minimal manual intervention such as change of an import line or copy-paste errors, or removal of input or print lines	Code correctly implements all algorithms, meets all required functionality with the exception of minor, rare test cases	Inputs and outputs formatted as per the specification both in terms of content and type, some formatting issues may be present (e.g. incorrect string text), correct file formats and names	Code uses structures and modules discussed in class with minor additions from outside sources (e.g. string formatting) appropriately cited; variable names meaningful; commented as appropriate
C	Code does not execute without minor manual intervention such as correcting indentation or terminating parentheses or a single syntax error	Code contains logical errors and fails subset of both rare and common test cases, overall algorithmic structure reflects required functionality but implementation does not meet standards	Inputs and outputs have major formatting issues, files incorrectly named but file formats correct	Code uses some structures and modules discussed in class but largely taken from outside sources with appropriate citations; most variable names meaningful
D	Code does not execute without major manual intervention such as changing variable names, correcting multiple syntax errors, algorithmic changes (e.g. infinite loops)	Code contains major logical errors and fails majority of test cases, lack of cohesive algorithmic structure, minimal functionality	Inputs and outputs have major formatting issues, files incorrectly named but file formats correct	Code uses structures and modules largely taken from outside sources with citations; variable names and code structure hard to decipher
F	Code requires extensive manual intervention to execute	Code fails all test cases and has no functionality to solve the proposed problem	Inputs and outputs have major formatting issues, files in incorrect formats which cannot be graded	Code uses structures and modules from outside sources only with no citations; variable names and code structure hard to decipher; code written in other programming languages or Python 2.7