CSCI 141 - Spring 2020
Final Exam
Due Date: Monday, July 8th

Stop work at 5:30pm PST, Turn in by 5:40 PST

Overview

For this final exam, you'll work collaboratively to create a program which helps with self-care in difficult times. This program is based off of 'Everything is Awful and I'm Not OK', a guide originally published at https://eponis.tumblr.com/post/113798088670/everything-is-awful-and-im-not-okay-questions-to . Someone made a minimally interactive version at https://philome.la/jace_harr/you-feel-like-shit-an-interactive-self-care-guide/play/index.html - your job is to make a version with 9 different features, as described below.

Apologies for the occasional swear words in the original sources. It is recommended you use cleaner language for the version you turn in so that it can be shared with a wider age range, but you won't get graded down if you don't, as long as the language is respectful and not offensive.

The original guide is released under a Creative Commons Attribution license. Cite it in your comments, and put in the comments that your portion of the code is licensed under a Creative Commons Share Alike Attribution Non-Commercial license.

It is recommended that you split the tasks among the different breakout rooms, however, you can as a class decide how you wish to organize the work.

If you choose to add additional features to any task, be sure it doesn't conflict with how the task will integrate with the other pieces (i.e., don't add new things to the write out of the .csv unless you coordinate that with the team which reads in the .csv).

Sample files and code has been provided so each task can be independently created. It is recommended that you allocate sufficient time within the two hours to combine the different pieces of code into a final solution. How you choose to do this is up to you. You could, for example, have each group contribute a file and 'include' these files into a main program. Or, you could copy and paste into a single file.

For each task, the team working on it can put their names, github, or linkedin link in the comments if they wish. After the exam has been graded, I'll post the file and share it publicly; only names and information in the comments will be included in the credits as authors.

Some tasks need a list of 16 topic keys, they are: hydrated, food, shower, stretch, nice, music, cuddle, therapist, meds, clothes, sleep, getitdone, selfie, plan, recover, wait.

1 Main Prompt

1.1 Description

Write a main prompt which allows the user to choose an option.

1.2 Checklist of things to implement

1. Ask the user to enter 1 for ‘Random Advice’, 2 for ‘Log Status’, and so on.
2. Call the associated function for the item the user chooses (get the function names from the other groups).
3. If the user enters an invalid option, tell them so.
4. After the chosen function has completed, use a loop to ask the user again.
5. If the user enters ‘0’, exit the program.
6. Include good in-line comments and doc strings.
7. Coordinate integrating the pieces from the other groups as they are ready. Note this task could be quite time consuming and tricky, so start coordinating with other groups early (i.e., how will they share their work with you? When do you need it by? What if they aren’t done? etc)
8. Once done, you can add additional features, or help other groups with checking for bugs.

2 Random Advice

2.1 Description

Write a function which offers the user a random piece of self-care advice.

2.2 Checklist of things to implement

1. Make a list with self-care at least 5 pieces of advice in it. Each piece of advice should be at least 2 sentences: One saying what to do, and at least one giving details or description of why it works. You can draw on the text from the links listed above, or write your own. You can include as many pieces of advice as you like.
2. Don’t copy text word from word from other sources than the ones in the ‘overview’ unless you cite the source (the name of the person who said it and where they said it is sufficient).
3. Ask the user if they’d like the advice sentence by sentence or as a paragraph.
4. Using the modality specified in the previous item, print out the advice.
5. Include good in-line comments and doc strings.
6. Once done, you can add additional features, or help other groups with checking for bugs.

3 Log Status

3.1 Description

Ask the user the 16 questions specified in the sources in ‘overview’, and write the answers to a csv file.

3.2 Checklist of things to implement

1. Ask the 16 questions from the source in ‘overview’. Use a list to store the text strings, and a loop to iterate over them.
2. The user will answer ‘yes’ or ‘no’ to each of the questions.
3. If the user enters bad input, tell them and ask the same question again.
4. Save the user input as a comma separated string. Write the string to a file called ‘log_status.csv’ stored in the same folder. An example log_status.csv has been provided for you to test with.
5. If there is already a ‘log_status.csv’ file, don’t delete the existing data, but write to the end of the file. (Hint: a couple different ways to do this, such as read in the original data to a variable, and write it back out, or use the ‘a’ option to append to the file. You can solve this any way you like though).
6. Include good in-line comments and doc strings.
7. Once done, you can add additional features, or help other groups with checking for bugs.

4 Read Status Log

4.1 Description

Read in the log_status.csv file and return the data as a list of dictionaries, for use by other functions.

4.2 Checklist of things to implement

1. An example log_status.csv has been provided for you to test with.
2. Read in the log_status.csv and save each line as a list of dictionaries. An example list of dictionaries has been provided, use the same keys.
3. Return this list of dictionaries.
4. Include good in-line comments and doc strings.
5. Once done, you can add additional features, or help other groups with checking for bugs.

5 Predict Topic

5.1 Description

Based on a list of dictionaries of previous log entries, choose the item with the fewest 'yes' answers and print out the key.

5.2 Checklist of things to implement

1. Take a list of dictionaries as a parameter. An example list of dictionaries has been provided for testing.
2. Calculate which item has the fewest 'yes' answers. This represents the item the user struggles with the most. Break ties as you wish, but document it.
3. Print out the key of the item from the task above.
4. Include good in-line comments and doc strings.
5. Once done, you can add additional features, or help other groups with checking for bugs.

6 Targeted Advice

6.1 Description

Given a topic key, print out advice for that topic.

6.2 Checklist of things to implement

1. Make a list of at least one piece of advice for each of the 16 topics from the source listed in 'overview'. Each piece of advice should be at least 2 sentences: One saying what to do, and at least one giving details or description of why it works. You can draw on the text from the links listed above, or write your own.
2. Don't copy text word from word from sources other than the ones in the 'overview' unless you cite the source (the name of the person who said it and where they said it is sufficient).
3. Ask the user what topic they would like advice on.
4. If the user enters something other than one of the 16 topic keys, tell them so, list the 16 topic keys, and ask them again.
5. Ask the user if they'd like the advice sentence by sentence or as a paragraph.
6. Using the modality specified in the previous item, print out the advice.
7. Include good in-line comments and doc strings.
8. Once done, you can add additional features, or help other groups with checking for bugs.

7 Log Good Memory

7.1 Description

Ask the user to remember a good memory or something they are grateful for, and write it to a file.

7.2 Checklist of things to implement

1. Ask the user for a good memory or something they are grateful for.
2. If the user doesn't enter input, tell them and ask the same question again. Write the user's answer to a file called 'log_good.txt' stored in the same folder. The only newline should be at the end of the entry. An example log_good.txt has been provided for you to test with.
3. If there is already a 'log_good.txt' file, don't delete the existing data, but write to the end of the file. (Hint: a couple different ways to do this, such as read in the original data to a variable, and write it back out, or use the 'a' option to append to the file. You can solve this any way you like though).
4. Include good in-line comments and doc strings.
5. Once done, you can add additional features, or help other groups with checking for bugs.

8 Remind Good Memory

8.1 Description

Drawing from the log_good.txt, remind the user of a random good memory or something they are grateful for.

8.2 Checklist of things to implement

1. An example log_good.txt has been provided for you to test with.
2. Read in the log_good.txt and save each line in a list. Note that each entry is separated by a new line.
3. Print out a random item of this list.
4. If the list is empty, tell the user to enter items from the main menu.
5. Include good in-line comments and doc strings.
6. Once done, you can add additional features, or help other groups with checking for bugs.

9 Make Art

9.1 Description

Let the user control a turtle to draw art, then print out the list of directions to re-create the art.

9.2 Checklist of things to implement

1. Create a turtle
2. Tell the user to press ‘w’ to move the turtle up, ‘s’ to move the turtle down, ‘a’ to move the turtle left, ‘d’ to move the turtle right, or ‘q’ to quit.
3. If the user enters invalid input, tell them and ask again.
4. Continue to ask the user until they enter ‘q’ to quit.
5. If the user enters a direction, move the turtle that direction, drawing a line. You can choose how far the turtle moves and what color they draw in.
6. Save the sequence of directions in a comma separated string.
7. When the user quits, print out the direction string, and then return it. An example direction string has been provided in the draw.csv file.
8. Include good in-line comments and doc strings.
9. Once done, you can add additional features, or help other groups with checking for bugs.

10 Share Art

10.1 Description

User can either create art and save it to a file, or read from a file with a direction string in it, and have the turtle re-draw that art.

10.2 Checklist of things to implement

1. Create a turtle
2. Ask user if they want to save or load.
3. If they want to save, call ’share_art’ and save the return value to draw.csv. A skeleton of ’share_art()’ has been provided with a sample return value. Overwrite what is in draw.csv if there is already data in the file.
4. If the user wants to load, read in the text from draw.csv file saved in the same directory. A sample draw.csv has been provided.

5. For each direction in draw.csv, have the turtle draw: W to move the turtle up, S to move the turtle down, A to move the turtle left, D to move the turtle right, or Q to quit.

6. The length of the each piece and the color is up to you (perhaps coordinate with the 'Make art' group for this part).

7. Include good in-line comments and doc strings.

8. Once done, you can add additional features, or help other groups with checking for bugs.

Submission

Upload `de-awfulizer.py` to Canvas and fill in quiz closing quiz about how it went. Each person should submit a solution. Submissions will be cross-referenced with attendance.

Each task will be graded on this rubric:

Rubric

Submission Mechanics (2 points)	
File called `de-awfulizer.py` is submitted to Canvas	2
Code Style and Clarity (3 points)	
Comment at the top with author/date/description	.5
Comments throughout code clarify any nontrivial code sections	.5
Variable and function names are descriptive	.5
Helper functions are used to keep functions no longer than about 30 lines of code (not counting comments and blank lines)	.5
The main sources listed in 'overview' and any additional sources are cited in the comments	.5
No global variables are referenced from within functions	.5
Correctness (5 points)	
Each task as described in the write up is completed correctly	5
Total	10 points