CECS 277 – Lab 5 – Classes & Objects

Task List

Create a program that maintains a task list for the user. The user should be able to view the current task, mark the current task complete, postpone the current task, or to add a new task. The program will read the list from a file ('tasklist.txt') when the program begins and then store the updated list by overwriting the old contents when the user quits the program.

Create a Task class (created in a separate file named 'task.py'):

Attributes:

1. `description` – string description of the task.
2. `date` – due date of the task. A string in the format: MM/DD/YYYY
3. `time` – time the task is due. A string in the format: HH:MM

Methods:

1. `__init__(self, desc, date, time)` – assigns the parameters to the attributes.
2. `get_description(self)` – returns the task's description string.
3. `__str__(self)` – returns a string used to display the task's information to the user in the format: description – Due: date at time.
4. `__repr__(self)` – returns a string used to write the task's information to the file in the format: task,date,time.
5. `__lt__(self, other)` – returns true if the self task is less than the other task. Compare by year, then month, then day, then hour, then minute, and then the task description in alphabetical order.

Main file ('main.py') – create the following functions:

1. `main_menu()` – displays the main menu and returns the user's valid input.
2. `read_file()` – Open the file ('tasklist.txt') and read in each of the tasks. Each line consists of the task description, due date, and time separated by commas. Construct a task object from each line and add it to a list. Return the filled task list.
3. `write_file(tasklist)` – passes in the list of tasks that will be written to the file ('tasklist.txt'). Iterate through the list of tasks and write each one to the file using the Task's repr() method (ie. description, date, and time separated by commas).
4. `get_date()` – prompts the user to enter the month, day, and year. Valid years are 2000-2100, valid months are 1-12, and valid days are 1-31 (no need to verify that it is a correct day for the month (ie. Feb 31st is valid)). Return the date in the format: MM/DD/YYYY. If the inputted month or day is less than 10, then add a leading 0 to format it correctly.
5. `get_time()` – prompts the user to enter the hour (military time) and minute. Valid hours are 0-23 and valid minutes are 0-59. Return the date in the format: HH:MM. If the inputted hour or minute is less than 10, then add a leading 0 to format it correctly.

Main Function – When the program starts, read in the contents of the file and store them in a sorted list. Repeatedly display the number of tasks and then prompt the user to choose from the following options until they quit the program:

1. Display Current Task – display the first task (which should be at the beginning of the list). If there are no tasks, then display a message that says all their tasks are complete.

2. Mark Current Task Complete – Display the current task, remove it and then display the new current task. If there are no tasks, then display a message.
3. Postpone Current Task – Display the current task and then prompt the user to enter a new date and time. Remove the task from the list and construct a new task using the old description with the updated due date and time. Add it back to the list and resort it. If there are no tasks, then display a message.
4. Add New Task – Prompt the user to enter a new task description, due date, and time. Construct the task using the user's input and add it to the list and resort it.
5. Save and Quit – write the contents of the task list back to the file (overwrite the old contents, do not append) and then end the program.

Example Output:

```
-Tasklist-
You have 2 tasks.
1. Display current task
2. Mark current task complete
3. Postpone current task
4. Add new task
5. Save and quit
Enter choice: 1
Current task is:
Submit Resume and Application -
Due: 12/05/2024 at 12:00

-Tasklist-
You have 2 tasks.
1. Display current task
2. Mark current task complete
3. Postpone current task
4. Add new task
5. Save and quit
Enter choice: 4
Enter a task: Do Essay
Enter due date:
Enter month: 12
Enter day: 1
Enter year: 2024
Enter time:
Enter hour: 14
Enter minute: 00

-Tasklist-
You have 3 tasks.
1. Display current task
2. Mark current task complete
3. Postpone current task
4. Add new task
5. Save and quit
Enter choice: 1
Current task is:
Do Essay - Due: 12/01/2024 at 14:00

-Tasklist-
You have 3 tasks.
1. Display current task
2. Mark current task complete
3. Postpone current task
4. Add new task
5. Save and quit
Enter choice: 2
Marking current task as complete:
Do Essay - Due: 12/01/2024 at 14:00
New current task is:
Submit Resume and Application -
Due: 12/05/2024 at 12:00

-Tasklist-
You have 2 tasks.
1. Display current task
2. Mark current task complete
3. Postpone current task
4. Add new task
5. Save and quit
Enter choice: 3
Postponing task:
Submit Resume and Application -
Due: 12/05/2024 at 12:00

Enter new due date:
Enter month: 12
Enter day: 10
Enter year: 2024
Enter new time:
Enter hour: 8
Enter minute: 00

-Tasklist-
You have 2 tasks.
1. Display current task
2. Mark current task complete
3. Postpone current task
4. Add new task
5. Save and quit
Enter choice: 1
Current task is:
```

```
Submit Resume and Application -
Due: 12/10/2024 at 08:00

-Tasklist-
You have 2 tasks.
1. Display current task
2. Mark current task complete
3. Postpone current task
4. Add new task
5. Save and quit
Enter choice: 5
Saving and exiting.
```

Notes:

1. Please place your name, date, and a brief description of the program in a comment block at the top of your program (main.py).
2. Use the check_input module to check the user's input for invalid values.
3. Do not add any additional attributes or methods to the class.
4. Please do not use the datetime, date, or time classes (we're creating our own).
5. Do not modify the methods or functions defined above (ie. same name and parameters).
6. You may create additional functions in main.py as needed.
7. Do not create any global variables or use attributes globally (use get_description instead).
8. Please follow the Coding Standards when writing your program.
9. Use docstrings to document your class, its attributes, methods, and main functions. Add brief comments in your program to describe sections of code.
10. Keep a spare copy of the tasklist.txt file handy. When you're testing your program, you will continuously be overwriting the contents and you'll want to replace it to test it again.
11. Thoroughly test your program before submitting:
    a. Make sure the file is read in properly, tasks are constructed and stored in the list (ie. it is not a list of strings, it is a list of task objects).
    b. Make sure that the tasks are sorted in ascending order. The __lt__ method should sort by year, and if those are the same, it should sort by month, then day, then hour, then minute, then if all of those are the same, then sort the description.
    c. Make sure that you display the total number of tasks to complete.
    d. Make sure that the current task is the soonest due date and time.
    e. Make sure that the current task is removed when marking it complete.
    f. Make sure to use str() when writing a task to the console, and repr() when writing a task to the file. Do not use the double underscores to call these methods.
    g. Make sure that when you're postponing, the current task is removed, and then a new task is constructed using the old description, and the new task is added.
    h. Make sure that the program does not crash when viewing, completing, postponing, saving an empty task list, or rerunning the program after saving.
    i. Make sure to error check all user input: Main menu: 1-5, Month: 1-12, Day: 1-31, Year: 2000-2100, Hour: 0-23, Minute: 0-59.
    j. Make sure the list is resorted after adding or postponing a task.
    k. Make sure that you write to the file in the same format (ie. comma separated with no spaces after the commas (spaces are ok in the description)).
    l. Avoid using commas when entering a description (I won't specifically test for this, but it will mess up your file).

Task List Rubric – Time estimate: 3-4 hours

Task List **10 points**	Correct. 2 points	A minor mistake. 1.5 points	A few mistakes. 1 point	Several mistakes. 0.5 points	No attempt. 0 points
Task class: 1. Created in a separate file. 2. Has attributes description, date, time. 3. Has methods: __init__, __str__, __repr__, __lt__, and get_description. 4. __lt__ compares the self and other tasks using year, month, day, hour, min, description. 5. Attributes are only accessed through the methods and not directly.					
Read and write functions: 1. Reads in file, create tasks, and adds them to the list of tasks. 2. Returns list of all tasks. 3. Write function passes in list of tasks. 4. Loops through task list and uses repr to write each task to the file.					
Get date and time functions: 1. Prompts user for month, day, year. 2. Returns valid date in correct format. 3. Prompts user for hour and minute. 4. Returns valid time in correct format.					
Main and main_menu functions: 1. Displays menu, prompts user, and returns valid menu input. 2. Reads in all tasks from file. 3. Displays current task (soonest task). 4. Add task, mark complete and postpone work correctly. 5. List is kept sorted. 6. Save and quit writes all tasks back to file (does not append). 7. Repeats until user quits. 8. Does not crash when re-executed.					
Code Formatting: 1. Code is in functions. 2. Correct spacing. 3. Meaningful variable names. 4. No global variables or accessing attributes directly. 5. Correct documentation.					