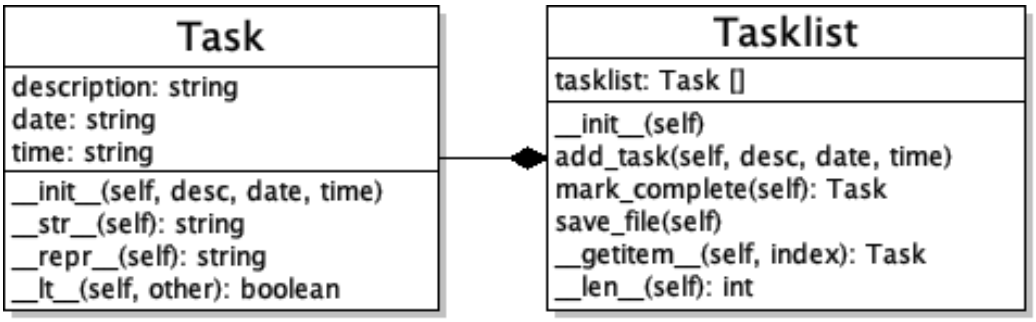
CECS 277 – Lab 6 – Class Relationships

Task List

Create a program that maintains a task list for the user. The user should be able to view the current task, list all of the tasks, mark the current task complete, 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)` – assign the parameters to the attributes.
2. `__str__(self)` – returns a string used to display the task's information to the user.
3. `__repr__(self)` – returns a string used to write the task to the file.
4. `__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 by alphabetical order.

Create a Tasklist class (created in a separate file named 'tasklist.py'):

Attributes:

1. `tasklist` – a list of Task objects.

Methods:

1. `__init__(self)` – read in the list of tasks from the file and store them in the tasklist by opening the file, reading in each line that consists of the task description, due date, and time separated by commas, then construct the Task object and appending it to the list, then sort the list.
2. `add_task(self, desc, date, time)` – construct a new task using the parameters, append it to the tasklist, and then sort the list.
3. `mark_complete(self)` – remove the current task from the tasklist.
4. `save_file(self)` – write the contents of the tasklist back to the file using the Task's __repr__ method (description, date, and time separated by commas).
5. `__getitem__(self, index)` – return the Task from the list at the specified index.
6. `__len__(self)` – return the number of items in the tasklist.

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

1. `main_menu()` – displays the main menu and returns the user's valid input.
2. `get_date()` – prompts the user to enter the year, month, and day. Valid years are 2000-3000, 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 0 to format it correctly.
3. `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 0 to format it correctly.

When the program starts, construct a Tasklist object. 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 to be completed. If there are no tasks, then display a message that says all their tasks are complete.
2. Display all tasks – display each of the tasks in the task list. If there are no tasks, then display a message that says their tasks are complete.
3. 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.
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-
Tasks to complete: 2
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Save and quit
Enter choice: 1
Current task is:
Submit Resume and Application
- Due: 12/05/2023 at 12:00

-Tasklist-
Tasks to complete: 2
1. Display current task
2. Display all tasks
3. Mark current task complete
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: 2023
Enter time:
Enter hour: 14
Enter minute: 00

-Tasklist-
Tasks to complete: 3
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Save and quit
Enter choice: 1
Current task is:
Do Essay - Due: 12/01/2023 at
14:00

-Tasklist-
Tasks to complete: 3
1. Display current task
2. Display all tasks
3. Mark current task complete
```

```
4. Add new task
5. Save and quit
Enter choice: 2
Marking current task as
complete:
Do Essay - Due: 12/01/2023 at
14:00
New current task is:
Submit Resume and Application
- Due: 2022/12/05 at 12:00

-Tasklist-
Tasks to complete: 2
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Save and quit
Enter choice: 2
Tasks:
Submit Resume and Application
- Due: 12/05/2023 at 12:00
Book Flight to New York - Due
01/10/2024 at 23:59

-Tasklist-
Tasks to complete: 2
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Save and quit
Enter choice: 5
```

Notes:

1. You should have 4 different files: task.py, tasklist.py, check_input.py, and main.py.
2. Please do not use the datetime, date, or time classes (we're using our own values).
3. Check all user input (other than the task description using the get_int_range function.
4. Please do not create any extra attributes, methods, or parameters. You may create additional functions in main.py as needed.
5. Please do not create any global variables or use the attributes globally. Only access the attributes using the class's methods.
6. Use docstrings to document the class, each of its methods, and the functions in main.py. See the lecture notes and the Coding Standards reference document for examples.
7. Please place your name, date, and a brief description of the program in a comment block at the top of your program (main.py). Place brief comments throughout your code.
8. Avoid using commas when entering a description (I won't specifically test for this, but it can mess up your file).
9. 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.
10. Thoroughly test your program before submitting:
    a. Make sure the file is read in properly, tasks are constructed and stored in the list.
    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 by the description.
    c. Make sure that you display the total number of tasks to complete.
    d. Make sure that the current task is the lowest due date and time.
    e. Make sure that the current task is removed when marking it complete.
    f. Make sure that the program does not crash when viewing, completing, or saving an empty task list, or when reading in the file on subsequent runs.
    g. Make sure to error check all user input: Main menu: 1-5, Year: 2000-2100, Month: 1-12, Day: 1-31, Hour: 0-23, Minute: 0-59.
    h. Make sure the list is resorted after adding a task.

i. 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)).

Task List Rubric – Time estimate: 4-5 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: desc, date, time. 3. Has methods: __init__, __str__, __repr__, and __lt__. 4. __lt__ compares tasks correctly.					
Tasklist class: 1. Created in a separate file. 2. Has tasklist attribute. 3. Has methods: __init__, add_task, mark_complete, save_file, __getitem__, and __len__ 4. __init__ reads from file, constructs Task objects, and puts them in the list. 5. save_file writes tasklist to file using __repr__ (same format as before).					
Functions of main: 1. Prompts user for input. 2. Validates user input. 3. Returns input (get_date, get_time formats the user's input).					
Main Function: 1. Constructs tasklist object. 2. Prompts user with menu. 3. Correctly displays current task. 4. Correctly displays all tasks (sorted). 5. Correctly marks task complete. 6. Correctly adds a new task. 7. Saves file when user quits.					
Code Formatting: 1. Correct spacing. 2. Meaningful variable names. 3. No exceptions thrown. 4. No global variables or accessing attributes directly. 5. Correct documentation.					