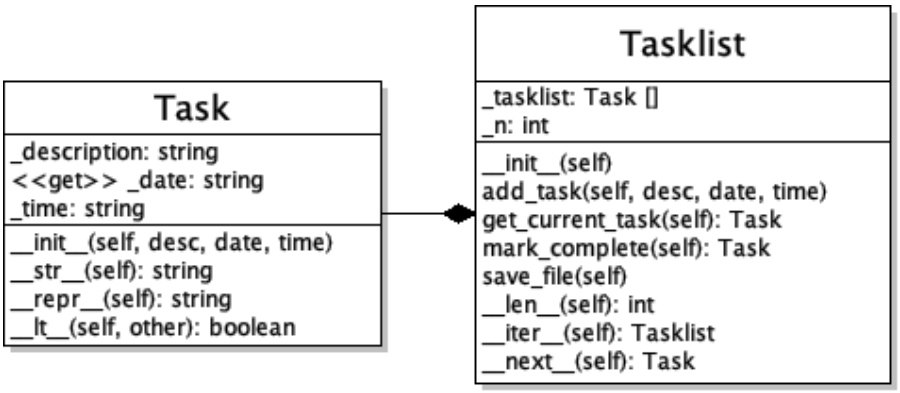
CECS 277 – Lab 12 – Iterator

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, search by date, 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.
2. `n` – a counter for the iterator (not initialized in the init).

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, constructing the Task object, appending it to the list, and then sorting 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. `get_current_task(self)` – return the task at the beginning of the list.
4. `mark_complete(self)` – remove and return the current task from the tasklist.

5. `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).
6. `__len__(self)` – return the number of items in the tasklist.
7. `__iter__(self)` – initialize the iterator attribute `n` and return self.
8. `__next__(self)` – iterate the iterator one position at a time. Raise a StopIteration when the iterator reaches the end of the tasklist, otherwise return the Task object at the iterator's current position.

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 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 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 begins, 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 – displays the first task using Tasklist's get_current_task method. If there are no tasks, then display a message that says all their tasks are complete.
2. Display all tasks – displays a numbered list of all the tasks in sorted order. It should be properly using the iterator that you created in the Tasklist class.
3. Mark current task complete – Displays the current task, removes it, and then displays the new current task by calling Tasklist's mark_complete method. If there are no tasks, then display a message.
4. Add new task – Prompts the user to enter a new task description, due date, and time, and adds the new task to the list by calling the Tasklist's add_task method.
5. Search by date – Prompts the user to enter a date (MM/DD/YYYY). Use the iterator to loop through the list, if the task's date is the same as the user's, then display the task. It should display all tasks with the matching date (not just the first one).
6. Save and quit – saves the updated contents of the list back to the file (does not append) and then quits the program.

Example Output:

```
-Tasklist-
Tasks to complete: 7
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Search by date
6. Save and quit
Enter choice: 4
Enter a task: Buy Flowers
Enter due date:
Enter month: 11
Enter day: 15
Enter year: 2023
Enter time:
Enter hour: 2
Enter minute: 00

-Tasklist-
Tasks to complete: 8
1. Display current task
2. Display all tasks
```

```
3. Mark current task complete
4. Add new task
5. Search by date
6. Save and quit
Enter choice: 1
Current task is:
Do Math Homework - Due:
11/05/2023 at 12:00

-Tasklist-
Tasks to complete: 8
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Search by date
6. Save and quit
Enter choice: 3
Marking current task as
complete:
Do Math Homework - Due:
11/05/2023 at 12:00
New current task is:
Write Email to Supervisor -
Due: 11/13/2023 at 12:00

-Tasklist-
Tasks to complete: 7
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Search by date
6. Save and quit
Enter choice: 5
Enter date to search:
Enter month: 11
Enter day: 15
Enter year: 2023
Tasks due on 11/15/2023:
1. Buy Flowers - Due:
11/15/2023 at 02:00
2. Do English Assignment -
Due: 11/15/2023 at 12:00
3. Babysit for Neighbor Kids
- Due: 11/15/2023 at 20:00

-Tasklist-
Tasks to complete: 7
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Search by date
6. Save and quit
Enter choice: 2
Tasks:
1. Write Email to Supervisor
– Due: 11/13/2023 at 12:00
2. Buy Flowers - Due:
11/15/2023 at 02:00
3. Do English Assignment -
Due: 11/15/2023 at 12:00
4. Babysit for Neighbor Kids
- Due: 11/15/2023 at 20:00
5. Haircut - Due: 11/20/2023
at 15:00
6. Do Grocery Shopping - Due:
11/21/2023 at 12:00
7. Clean House - Due:
11/26/2023 at 12:00

-Tasklist-
Tasks to complete: 7
1. Display current task
2. Display all tasks
3. Mark current task complete
4. Add new task
5. Search by date
6. Save and quit
Enter choice: 6
Saving List...
```

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. 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.
4. Check all user input (other than the task description using the get_int_range function.
5. Please do not create any extra attributes, methods, or parameters. You may create additional functions in main.py as needed.
6. Please do not create any global variables or use the attributes globally. Only access the attributes using the class's methods.
7. 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.
8. 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.
9. 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: 1-6, 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)).
    j. Make sure that options 1 and 5 are properly using the iterator that you wrote.
    k. 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: 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 (separate file): 1. Has attributes: desc, date, time. 2. Has methods: __init__, __str__, __repr__, __lt__, and date property. 3. __lt__ compares tasks correctly.					
Tasklist class (separate file): 1. Has methods: __init__, add_task, mark_complete, get_current_task, save_file, and __len__ 2. __init__ reads from file, constructs Tasks, adds it to the list, then sorts. 3. save_file writes tasklist to file using __repr__ (same format).					
Iterator (in Tasklist class): 1. Has iter and next methods. 2. iter initializes n and returns iterator. 3. next iterates through the tasklist one item at a time and returns the task. 4. Correctly raises StopIteration when it reaches the end of the list.					
Main (separate file): 1. Constructs Tasklist object. 2. Has menu, get_date, get_time funcs. 3. Correctly displays current task. 4. Correctly displays all tasks (sorted) using the iterator. 5. Correctly marks tasks complete. 6. Correctly adds a new task. 7. Correctly searches for a date using the iterator and finds all matches. 8. Saves file when user quits. 9. Doesn't crash when removing last task or when re-opening file. 10. Error checks all user input.					
Code Formatting: 1. Correct documentation. 2. Meaningful variable names. 3. No exceptions thrown. 4. No global variables or accessing attributes directly. 5. Correct spacing.					