

Deja Monet

Randal Root

Foundations of Python

17 August 2022

[GitHub](#)

Assignment 06: Functions & Classes

Introduction

In this assignment, I will review defining functions and classes. I will also review installing libraries for Jupyter Lab in Anaconda Prompt, splitting classes between cells in Jupyter Lab, and calling Jupyter .ipynb files in Anaconda Prompt.

Completing the Assignment: Writing the Code

In this assignment, we returned to using functions, so I returned to using Jupyter Lab as my IDE. I first split all of the pre-defined functions into different cells in the .ipynb file as to better organize my script. When initializing all of the cells, I ran into a problem where the functions were not recognized as part of their class, as they were split between cells. I researched this problem on Stack Overflow, and found that I could use the **jupyter dynamic class (jdc)** package to solve this (**Figure 1**).



Figure 1. An example from Stack Overflow of using the jdc module to split multiple functions from one class across cells in Jupyter Lab.

I installed this package in Anaconda Prompt by calling **pip install jdc** from the base environment, imported the package in my .ipynb file using **import jdc**, used the **%%add_to** function in order to add

functions from different cells to a class in a different cell, and was then able to initialize all of the pre-defined functions in different cells.

The first step was to load any existing data from the `ToDoFile.txt` in the working directory using the `read_data_from_file` function from the `Processor` class. For this step, I called the `exists` function from the `os.path` module to check if the `ToDoFile.txt` file exists in the working directory. If the file does exist, the data is loaded from the file and the user is alerted of this. If this file does not exist, the user is alerted there is no data to load and is returned to the main menu (**Figure 2**).

```
# Processing ----- #
class Processor():
    """Performs Processing tasks"""

    %%add_to Processor
    @staticmethod
    def read_data_from_file(file_name, list_of_rows):
        """
        Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """

        #check if file exists
        file_exists = exists(file_name)

        #if file exists, load data
        if(file_exists == True):
            list_of_rows.clear() #clear current data
            file = open(file_name, "r")
            for line in file:
                task, priority = line.split(",")
                row = {"Task": task.strip(), "Priority": priority.strip()}
                list_of_rows.append(row)
            file.close()
            print("Data loaded from file!")
            return list_of_rows

        #if file does not exist, alert user and return to main menu
        else:
            print("No data to load from file!")
```

Figure 2. The function `read_data_from_file` loads data from the `ToDoFile.txt` file if this file exists in the working directory.

The next step was to display the current data and menu of choices to the user. I decided to alter this step such that the user is only shown the to-do list if there is data in the to-do list (**Figure 3**).

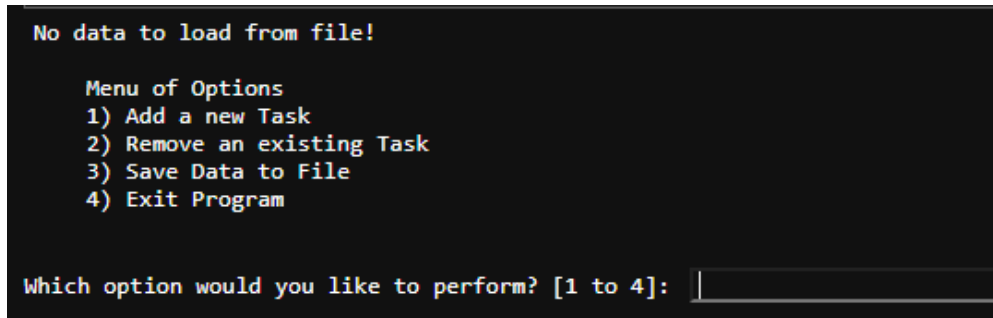
```
### Main Body of Script ----- #
"""Step 1: When the program starts, load data from ToDoFile.txt"""
Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst) # read data from file

"""Step 2: Display a menu of choices to the user"""
while (True):
    """Step 3: show current data"""
    #shows current data in the to-do list if there are items in the to-do list
    if(table_lst != []):
        IO.output_current_tasks_in_list(list_of_rows=table_lst)

    #shows main menu to user, returns user's choice from main menu
    IO.output_menu_tasks()
    choice_str = IO.input_menu_choice()
```

Figure 3. The `output_current_tasks_in_list` function is only called if there is data stored in the `table_lst` variable.

In this alteration, the user is only shown their current to-do list if there is data on the to-do list. I made this change by adding an if statement to the while loop in the main body of the script instead of altering the **output_current_tasks_in_list** function. In the example below, there is no ToDoFile.txt in the working directory, so the user is not shown their current data and is only shown the main menu (**Figure 4**).



```

No data to load from file!

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

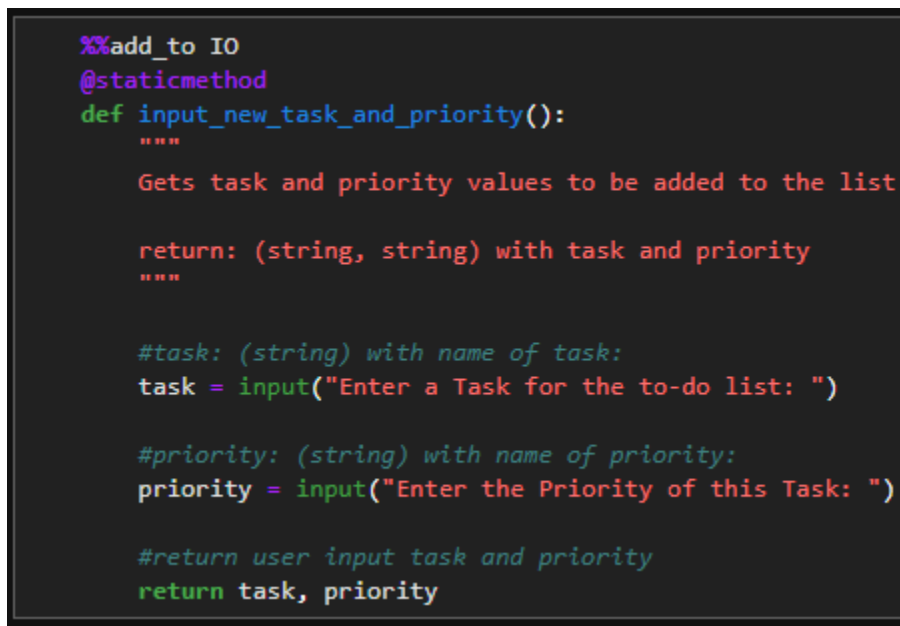
Which option would you like to perform? [1 to 4]: 

```

Figure 4. The user is not shown the "Here is your current to-do list" text if there is no data in the to-do list.

I also did not make any changes to the **output_menu_tasks** function or to the **input_menu_choice** function in the IO class.

The next step was to complete the code in order to process the user's choice from the main menu. The first choice is to add new data to the to-do list using the **input_new_task_and_priority** function and the **add_data_to_list** function from the IO and Processor classes, respectively. I first added code to retrieve the user input data (**Figure 5**).



```

%%add_to IO
@staticmethod
def input_new_task_and_priority():
    """
    Gets task and priority values to be added to the list

    return: (string, string) with task and priority
    """

    #task: (string) with name of task:
    task = input("Enter a Task for the to-do list: ")

    #priority: (string) with name of priority:
    priority = input("Enter the Priority of this Task: ")

    #return user input task and priority
    return task, priority

```

Figure 5. **input_new_task_and_priority** function retrieves user input data and then returns this data.

Then, in the **add_data_to_list** function, I added the user input task and priority to a dictionary row, then appended this row to the **list_of_rows** variable and returned the new **list_of_rows** (**Figure 6**).

```

%%add_to Processor
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    """
    Adds data to a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """

    #add user input items to dictionary row
    row = {"Task": str(task).strip(), "Priority": str(priority).strip()}

    #add dictionary row to python list/table
    list_of_rows.append(row)

    return list_of_rows

```

Figure 6. The user input data is added to a dictionary row, and this row is then appended to a table.

The next step was to allow the user to remove a task from the to-do list. I added a conditional if statement such that the script will only allow the user to choose an item to remove from the to-do list if there is data in the to-do list (**Figure 7**).

```

"""Step 4: Process user's menu choice"""
#add a new Task to the to-do list
if(choice_str.strip() == '1'):
    task, priority = IO.input_new_task_and_priority()
    table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
    continue #return to main menu

#remove an existing Task from the to-do list
elif(choice_str == '2'):
    #if the to-do list is empty, alert the user and return to main menu
    if(table_lst == []):
        print("There are no tasks on the to-do list! Please choose a different option:")

    #if there are tasks on the to-do list, allow the user to remove a task from the to-do list
    else:
        task = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
    continue #return to main menu

```

Figure 7. The user is alerted if they try to remove an item from the to-do list when the to-do list is empty.

If there is data on the to-do list, the user is able to remove an item. I also added code to the `input_task_to_remove` function by capturing the user data as the variable `str_task_remove` and returning this variable for use in the next function (**Figure 8**).

```

%%add_to IO
@staticmethod
def input_task_to_remove():
    """
    Gets the task name to be removed from the list

    :return: (string) with task
    """

    #identify item to remove from to-do list
    str_task_remove = input("Which task do you want to remove from the to-do list? ")

    return str_task_remove

```

Figure 8. The user input task to remove from the to-do list is captured as `str_task_remove`.

Next, I added data to the `remove_data_from_list` function in order to cycle through all of the tasks in the to-do list until a task matches the user input option, and is then removed. If all of the items in the to-do list are cycled through and a match is not found, the user is alerted that their input item was not found on the to-do list (Figure 9).

```

%%add_to Processor
@staticmethod
def remove_data_from_list(task, list_of_rows):
    """
    Removes data from a list of dictionary rows

    :param task: (string) with name of task:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """

    #cycle through entries in list_of_rows until the user input item
    #matches an item in the list of dictionary rows
    list_of_rows_position = 0
    for row in list_of_rows:
        #if user input item is found on the to-do list, it is removed from the to-do list
        if(row["Task"].lower() == task.lower()):
            #removes user input item from list
            del list_of_rows[list_of_rows_position]

            #alert user the item was removed from the list and return to main menu
            print(task + " has been removed from the to-do list.")
            break

        #if the list has been cycled through, and the user input item is not
        #the last item on the list: alert user the item was not found and return to main menu
        if(str(list_of_rows_position) == str(len(list_of_rows)-1) and row["Task"].lower() != task.lower()):
            print(task + " was not found on the to-do list!")
            break

        #if user input item has not been found, but there are more items on the list,
        #then continue to cycle through the list
        else:
            list_of_rows_position = list_of_rows_position+1

    return list_of_rows

```

Figure 9. If the user input task to remove is found, it is removed from the to-do list. If it is not found, the user is alerted and returned to the main menu.

I show this in practice below by attempting to remove an item when there is no data on the to-do list, then adding “wash dishes” and “do laundry” to the to-do list, trying to remove “vacuum” from the to-do list, and finally removing “wash dishes” (Figure 10).

```

No data to load from file!

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

There are no tasks on the to-do list! Please choose a different option:

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: wash dishes
Enter the Priority of this Task: 2
Here is your current to-do list:
wash dishes (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: do laundry
Enter the Priority of this Task: 1
Here is your current to-do list:
wash dishes (2)
do laundry (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? vacuum
vacuum was not found on the to-do list!
Here is your current to-do list:
wash dishes (2)
do laundry (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? wash dishes
wash dishes has been removed from the to-do list.
Here is your current to-do list:
do laundry (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 4

Goodbye!

```

Figure 10. The script alerts the user when they try to remove a task from the to-do list when there are no tasks on the to-do list.

The next step was to allow the user to save their data to a .txt file. In the **write_data_to_file** function, I used the pre-defined variables **file_obj** and **file_name_str** to create or open the **ToDoFile.txt** file, then used a for loop to cycle through all of the items in the **list_of_rows** table, and finally, alert the user when their data has been saved (**Figure 11**).

```

%%add_to Processor
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """
    Writes data from a list of dictionary rows to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """

    #ToDoFile.txt is identified as file_obj
    file_obj = open(file_name_str, "w")

    #write user input to-do items to ToDoFile.txt
    for row in list_of_rows:
        file_obj.write(row["Task"] + "," + row["Priority"] + "\n")

    #ToDoFile.txt is closed
    file_obj.close()

    #user is alerted that their data has successfully been saved
    print("Data has been saved!")

    return list_of_rows

```

Figure 11. A for loop is used to cycle through all of the user input to-do items and write them to a text file.

The last step was to exit the script, and I made no changes to this part of the code. With the script completed, I could begin testing the script in Jupyter Lab and Anaconda Prompt.

Completing the Assignment: Testing the Script

I began testing the script in Jupyter Lab. I started with no ToDoFile.txt existing in the working directory. I added “grocery shopping” and “iron clothes” to the to-do list as priority 1 and 2 (respectively), and attempted to remove “wash dishes” (**Figure 12**).

```
No data to load from file!

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: grocery shopping
Enter the Priority of this Task: 1
Here is your current to-do list:
grocery shopping (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: iron clothes
Enter the Priority of this Task: 2
Here is your current to-do list:
grocery shopping (1)
iron clothes (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? wash dishes
wash dishes was not found on the to-do list!
Here is your current to-do list:
grocery shopping (1)
iron clothes (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? iron clothes
iron clothes has been removed from the to-do list.
Here is your current to-do list:
grocery shopping (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program
```

Figure 12. Part 1/4 of testing the script in Jupyter Lab.

I then removed “iron clothes,” added “vacuum” as priority 2, saved the data to a file, and then exited the script (**Figure 13**).

```
Which task do you want to remove from the to-do list? iron clothes
iron clothes has been removed from the to-do list.
Here is your current to-do list:
grocery shopping (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: vacuum
Enter the Priority of this Task: 2
Here is your current to-do list:
grocery shopping (1)
vacuum (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 3

Data has been saved!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 4

Goodbye!
```

Figure 13. Part 2/4 of testing the script in Jupyter Lab.

I then ran the script again to check that data would be loaded correctly from the existing ToDoFile.txt file. I then removed “grocery shopping,” added “buy socks” as priority 1, and closed the script without saving the data (**Figure 14**).

```
Data loaded from file!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? grocery shopping
grocery shopping has been removed from the to-do list.
Here is your current to-do list:
vacuum (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: buy socks
Enter the Priority of this Task: 1
Here is your current to-do list:
vacuum (2)
buy socks (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 4

Goodbye!
```

Figure 14. Part 3/4 of testing the script in Jupyter Lab.

Lastly, I ran the script again to make sure that the data did not save to the text file, attempted to remove “buy socks,” and closed the script again without saving (**Figure 15**).

```

Data loaded from file!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? buy socks
buy socks was not found on the to-do list!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 4

Goodbye!

```

Figure 15. Part 4/4 of testing the script in Jupyter Lab.

I then needed to test the script in Anaconda Prompt. I began by changing the working directory to the directory where the script file was stored and converted the .ipynb file to a .py file using **jupyter nbconvert**. I tried calling the script using `python Assignment06.py`, but ran into the following error (Figure 16).

```

Anaconda Prompt (anaconda3)
(base) C:\Users\dejam>cd C:\PythonClass\Assignment06

(base) C:\PythonClass\Assignment06>jupyter nbconvert --to python Assignment06.ipynb
[NbConvertApp] Converting notebook Assignment06.ipynb to python
[NbConvertApp] Writing 9660 bytes to Assignment06.py

(base) C:\PythonClass\Assignment06>python Assignment06.py
Traceback (most recent call last):
  File "C:\PythonClass\Assignment06\Assignment06.py", line 24, in <module>
    import jdc #for use in Jupyter Lab: jupyter dynamic classes (jdc) module allows functions in a class
    to be spread across cells
  File "C:\Users\dejam\anaconda3\lib\site-packages\jdc\__init__.py", line 5, in <module>
    ip = get_ipython()
NameError: name 'get_ipython' is not defined

```

Figure 16. An error running the converted .py file using the python command in Anaconda Prompt.

In researching the **get_ipython** error, I learned that I should call the converted script using the **ipython** kernel instead of **python**. I tested this by calling the script using **ipython**, and when this worked, continued testing the script. I added “pay rent” and “organize desk” as priority 1 and 4 (respectively) (Figure 17).

```
(base) C:\_PythonClass\Assignment06>ipython Assignment06.py
Data loaded from file!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: pay rent
Enter the Priority of this Task: 1
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 1

Enter a Task for the to-do list: organize desk
Enter the Priority of this Task: 4
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)
organize desk (4)
```

Figure 17. Part 1/4 of testing the script in Anaconda Prompt.

Then, I saved the file and exited the script. (**Figure 18**).

```
Enter a Task for the to-do list: organize desk
Enter the Priority of this Task: 4
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)
organize desk (4)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 3

Data has been saved!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)
organize desk (4)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

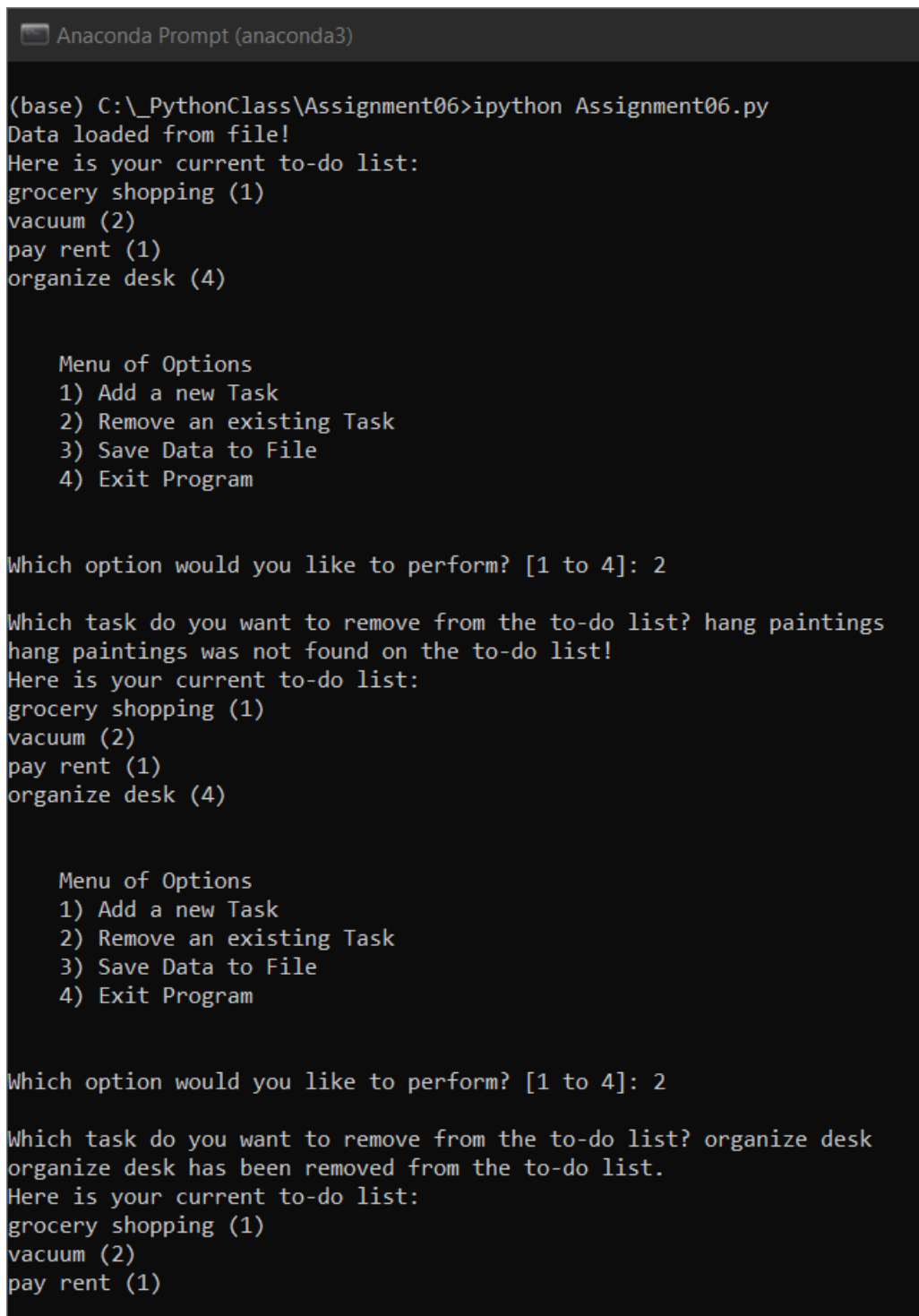
Which option would you like to perform? [1 to 4]: 4

Goodbye!

(base) C:\_PythonClass\Assignment06>
```

Figure 18. Part 2/4 of testing the script in Anaconda Prompt.

I then ran the converted .py script again with ipython, attempted to remove “hang paintings,” and removed “organize desk” from the to-do list (**Figure 19**).



```
Anaconda Prompt (anaconda3)

(base) C:\_PythonClass\Assignment06>ipython Assignment06.py
Data loaded from file!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)
organize desk (4)

    Menu of Options
    1) Add a new Task
    2) Remove an existing Task
    3) Save Data to File
    4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? hang paintings
hang paintings was not found on the to-do list!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)
organize desk (4)

    Menu of Options
    1) Add a new Task
    2) Remove an existing Task
    3) Save Data to File
    4) Exit Program

Which option would you like to perform? [1 to 4]: 2

Which task do you want to remove from the to-do list? organize desk
organize desk has been removed from the to-do list.
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)
```

Figure 19. Part 3/4 of testing the script in Anaconda Prompt.

I then saved the file and exited the script (**Figure 20**).

```
Which task do you want to remove from the to-do list? organize desk
organize desk has been removed from the to-do list.
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 3

Data has been saved!
Here is your current to-do list:
grocery shopping (1)
vacuum (2)
pay rent (1)

Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4]: 4

Goodbye!

(base) C:\_PythonClass\Assignment06>
```

Figure 20. Part 4/4 of testing the script in Anaconda Prompt.

With all of the errors resolved, the assignment was complete.

Summary

In this assignment, I reviewed functions, classes, and calling functions within classes. I also used Jupyter Lab modules in order to organize classes and functions, and learned how to call Jupyter files in Anaconda Prompt.