Name: Jane Shin
Date: November 23, 2022
Course: IT FDN 110 A Au 22: Foundations of Programming: Python

https://github.com/ks180337/IntroToProg-Python

Assignment 06 – Functions & Classes

Introduction

This document describes the steps taken in performing Assignment 06, where a Python script file is created, using an existing starter script, that manages a “To-do List”. The script uses a printed "menu" to guide the user through various options for the “To-do List”, including displaying current list, adding a new item, removing an existing item, saving data to the original file, and exiting the program. The scope of work is similar to that of Assignment 05, however, the script for Assignment 06 uses functions and classes to organize Data, Processing, and Presentation sections.

The "To-do List" file contains two columns of data, "Task" and "Priority." The columns are loaded into a Python dictionary object. Each dictionary object represents one row of data, and these rows are added to a Python list object to create a table of data.

This assignment requires updating existing template and code. New tools and concepts introduced in Module 6, such as functions and classes, as well as previously addressed concepts such as list, dictionary, reading and writing data from a file into a list/dictionary will be used in accomplishing this task.

Instructions

Module 6 covered a detailed application of the function and class concepts, delivered through course videos, a book chapter, and web pages. The course material delivered detailed explanation and examples of the following topics:

- Using functions to organize codes
- Understanding the usage of parameters, arguments, and return values within a function
- Distinguishing a global and a local variable
- Using classes to organize functions
- Using classes to program to a pattern called “Separations of Concerns”
- Using and familiarizing the debugging tools used in PyCharm
- Using GitHub for source control

Creating the Script

Header and Initial Comments

```python
# ------------------------------------------------------------------------
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoList.txt" into a python Dictionary.
#              Add each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# JShin,11.23.2022,Modified code to complete assignment 06
# ------------------------------------------------------------------------ #
```

Figure 1. Header and initial comments

- PyCharm was used to create and run the script. The assignment was saved as a .py file named "Assignment06.py" at the location according to a direction given in the instruction.
- Script header and comments – A script header was created per the examples given in the instruction to communicate the function and document changes of the script.
- The Change Log was updated to reflect the codes added to the original starter script to complete the assignment.

Declaration of Variables

```python
# Data ------------------------------------------------------------------- #
# Declare variables and constants
file_name_str = "ToDoList.txt"  # The name of the data file
file_obj = None  # An object that represents a file
row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priorit
table_lst = []  # A list that acts as a 'table' of rows
choice_str = ""  # Captures the user option selection
task_str = ""  # Captures the task input from functions to be processed in main bod
priority_str = ""   # Captures the priority input from functions to be processed in
```

Figure 2. Declaration of variables

- Using the "Separation of Concerns" concept, variables and constants are declared within the "Data" section along with comments explaining the purposes of each variable.
- Variables are organized in order of appearance.
- Variable from the original script were retained, and several new variables were added.

Overall Script Structure

```
23  # Processing  ------------------------------------------------------------ #
24  class Processor:...
83
84
85  # Presentation (Input/Output)  ------------------------------------------- #
86  class IO:...
145
146 # Main Body of Script  ------------------------------------------------- #
147
148 # Step 1 - When the program starts, Load data from ToDoFile.txt.
149 Processor.read_data_from_file( file_name=file_name_str, list_of_rows=table_lst)  # read file data
150
151 # Step 2 - Display a menu of choices to the user
152 while (True):
153     # Step 3 Show current data
154     IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the list/table
155     IO.output_menu_tasks()  # Shows menu
156     choice_str = IO.input_menu_choice()  # Get menu option
157
158     # Step 4 - Process user's menu choice
159     if choice_str.strip() == '1':...
163
164     elif choice_str == '2':...
168
169     elif choice_str == '3':...
173
174     elif choice_str == '4':...
```

Figure 3. Overall Script Structure

- Since the assignment was to update a pre-existing script, it was necessary to understand the overall structure and intent of the original starter script. Figure 3. shows a high-level flow of the starter script.
- The "Processing" section contains a class named **Processor**. This class consists of several functions that processes data such as reading from or writing to a file or a list.
- The "Presentation" section contains a class named **IO**. This class consists of several functions that handles inputs and outputs of data such as presenting a menu and obtaining a menu choice from the user.
- The details of each class and the functions belonging to the class are explained in later sections.
- The main body of the script will call upon the class and the functions to carry out the intent of the script.

The “Processing” Section – class Processor

```python
# Processing  ------------------------------------------------------------ #
class Processor:
    """  Performs Processing tasks """

    @staticmethod
    def read_data_from_file(file_name, list_of_rows):...

    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):...

    @staticmethod
    def remove_data_from_list(task, list_of_rows):...

    @staticmethod
    def write_data_to_file(file_name, list_of_rows):...
```

Figure 4. Overview of the “Processing” Section – class **Processor**

- The “Processing” section contains a class named **Processor**. The class **Processor** consists of (4) functions that perform processing tasks:
    - read_data_from_file
    - add_data_to_list
    - remove_data_from_list
    - write_data_to_file

The “Processing” Section – class Processor (cont.)

Function read_data_from_file

```python
    @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
        """
        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()
        return list_of_rows
```

Figure 5. Function **read_data_from_file**

- The **read_data_from_file** function was defined with two parameters, *file_name* and *list_of_rows.* It will read data from a file into a list of dictionary rows.

- It is a common practice to include a helpful header at the beginning of a function, which is known as *docstring*. PyCharm can display tooltips to show you a developer's notes (ctrl + q). This is common throughout the functions defined in the rest of the script.
- The **open()** function was used to open the file that contains initial data, "ToDoList.txt".
- "r" mode allows to read from a text file. If the file does not exist, Python will generate an error.
- The file is then assigned to the variable *file*.
- A **for** loop is chosen to loop through the data file, while reading each "line" of text data, split with a comma, from the file into string variables *task, priority*, respectively.
- The values of *task, priority* then were assigned to a dictionary object representing one row of data, named *row*, along with the keys "Task" and "Priority". Note the curly brackets indicating a dictionary object.
- **strip()** was used to remove any unnecessary space or carriage return.
- *row* is then appended to a list object named *list_of_rows*.
- The **close()** function was used to close the file.
- The **read_data_from_file** function then returns a list object named *list_of_rows* as its return value.

The "Processing" Section – class Processor (cont.)

Function add_data_to_list

```
    @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
        """
        row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
        list_of_rows.append(row)
        return list_of_rows
```

Figure 6. Function **add_data_to_list**

- The **add_data_to_list** function was defined with three parameters, *task, priority,* and *list_of_rows.* It will add data to a list of dictionary rows.
- Arguments passed through parameters *task* and *priority* are assigned to a dictionary object *row*, using curly brackets to indicate that it is a dictionary.
- **strip()** was used to remove any unnecessary space or carriage return.
- *row* is appended to a list object named *list_of_rows*.
- The **add_data_to_list** function then returns a list object named *list_of_rows* as its return value.

The "Processing" Section – class Processor (cont.)

Function remove_data_from_list

```python
    @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
        """
        for row in list_of_rows:
            if row["Task"].lower() == task.lower():
                list_of_rows.remove(row)
        return list_of_rows
```

Figure 7. Function **remove_data_from_list**

- The **remove_data_from_list** function was defined with two parameters, *task* and *list_of_rows.* It will remove data from a list of dictionary rows.
- A **for** loop is used to loop through *list_of_rows,* while comparing the argument to the stored value with key named "Task". If the *task* matches the value, the corresponding row of the *list_of_rows* is removed.
- The **remove_data_from_list** function then returns a list object named *list_of_rows* as its return value.

Function write_data_to_file

```python
    @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
        """
        file = open(file_name, "w")
        for row in list_of_rows:
            file.write(row["Task"] + "," + row["Priority"] + "\n")
        file.close()
        return list_of_rows
```

Figure 8. Function **write_data_to_file**

- The **write_data_to_file** function was defined with two parameters, *file_name* and *list_of_rows.* It will write data from a list of dictionary rows to a file.
- The **open()** function was used to open a file.
- The "w" mode allows to write to a text file. If the file already exists, its content is overwritten. If the file does not exist, it is newly created.
- The file is then assigned to the variable *file*.
- A **for** loop is used to loop through *list_of_rows*, while writing each "row" of dictionary into file.
- The **write()** method was used to write data stored in variables into the created file.
- After each row, a new line was created with "\n".
- The **close()** method was used to close the file.

- The **write_data_to_file** function then returns a list object named *list_of_rows* as its return value.

The "Presentation" Section – class IO

```python
# Presentation (Input/Output) ------------------------------------------ #
class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def output_menu_tasks():...

    @staticmethod
    def input_menu_choice():...

    @staticmethod
    def output_current_tasks_in_list(list_of_rows):...

    @staticmethod
    def input_new_task_and_priority():...

    @staticmethod
    def input_task_to_remove():...
```

Figure 9. Overview of the "Presentation" Section – class **IO**

- The "Presentation" section contains a class named **IO**. The class **IO** consists of (5) functions that perform input and output tasks:
    - output_menu_tasks
    - input_menu_choice
    - output_current_tasks_in_list
    - input_new_task_and_priority
    - input_task_to_remove

The "Presentation" Section – class IO (cont.)

Function output_menu_tasks

```python
    @staticmethod
    def output_menu_tasks():
        """  Display a menu of choices to the user

        :return: nothing
        """
        print('''
        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program
        ''')
        print()  # Add an extra line for looks
```

Figure 10. Function **output_menu_tasks**

- The **output_menu_tasks** function was defined with no parameters and no return value. It will display a menu of choices to the user.
- Menu of options are displayed using **print()** function.

The "Presentation" Section – class IO (cont.)

Function input_menu_choice

```python
    @staticmethod
    def input_menu_choice():
        """ Gets the menu choice from a user

        :return: string
        """
        choice = str(input("Which option would you like to perform? [1 to 4] - ")).
        print()  # Add an extra line for looks
        return choice
```

Figure 11. Function **input_menu_choice**

- The **input_menu_choice** function was defined with no parameters. It will obtain the menu of choices from the user.
- The choice obtained from user through **input()** function are converted to string data type, which will be used for the remainder of the script.
- The generated data was assigned to variable *choice* to be used in subsequent operations.
- The **input_menu_choice** function then returns a string variable named *choice* as its return value.

Function output_current_tasks_in_list

```python
    @staticmethod
    def output_current_tasks_in_list(list_of_rows):
        """ Shows the current Tasks in the list of dictionaries rows

        :param list_of_rows: (list) of rows you want to display
        :return: nothing
        """
        print("******* The current tasks ToDo are: *******")
        for row in list_of_rows:
            print(row["Task"] + " (" + row["Priority"] + ")")
        print("*****************************************")
        print()  # Add an extra line for looks
```

Figure 12. Function **output_current_tasks_in_list**

- The **output_current_tasks_in_list** function was defined with a parameter *list_of_rows,* and no return value. It will show the current tasks in the list.
- A **for** loop is used to loop through the nested list *list_of_rows,* while printing each "rows". The keys were used to retrieve the corresponding values from the dictionary objects, formatted with parentheses around the "Priority" for clarity.

The “Presentation” Section – class IO (cont.)

Function input_new_task_and_priority

```python
    @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 = str(input("Enter a task: ")).strip()
        priority = str(input("Enter the priority (High, Medium, Low): ")).strip()
        return task, priority
```

Figure 13. Function **input_new_task_and_priority**

- The **input_new_task_and_priority** function was defined with no parameters, and two string return values. It will obtain task and its priority to be added to the list from the user.
- **input()** function was used to prompt user to input the data required for program.
- The data obtained from user through **input()** functions are initially of string data type, which will be used for the remainder of the script.
- Obtained data was assigned to variables *task*, *priority,* respectively.
- **strip()** was used to remove any unnecessary space or carriage return.
- The **input_new_task_and_priority** function then returns string variables named *task*, *priority* as its return values.

Function input_task_to_remove

```python
    @staticmethod
    def input_task_to_remove():
        """  Gets the task name to be removed from the list

        :return: (string) with task
        """
        task = str(input("Enter a task to remove: ")).strip()
        return task
```

Figure 14. Function **input_task_to_remove**

- The **input_task_to_remove** function was defined with no parameters, and a string return value. It will obtain the task name to be removed from the list from the user.
- **input()** function was used to prompt user to input the data required for program.
- The data obtained from user through **input()** functions are initially of string data type, which will be used for the remainder of the script.
- Obtained data was assigned to a variable *task*.
- **strip()** was used to remove any unnecessary space or carriage return.
- The **input_task_to_remove** function then returns string variables named *task* as its return values.

Main Body of Script

```python
# 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 file data

# Step 2 - Display a menu of choices to the user
while (True):
    # Step 3 Show current data
    IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the list/table
    IO.output_menu_tasks()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

    # Step 4 - Process user's menu choice
    if choice_str.strip() == '1':  # Add a new Task
        task_str, priority_str = IO.input_new_task_and_priority()
        table_lst = Processor.add_data_to_list(task=task_str, priority=priority_str, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '2':  # Remove an existing Task
        task_str = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(task=task_str, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '3':  # Save Data to File
        table_lst = Processor .write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
        print("Data Saved!")
        continue  # to show the menu

    elif choice_str == '4':  # Exit Program
        print("Goodbye!")
        break  # by exiting loop
```

Figure 15. Main Body of Script

- Step 1
  - It loads data from the text file when the program starts. The format of **class.function** was used in calling **Processor.read_data_from_file** to read file data.
  - Variables *file_name_str* and *table_lst* were passed as arguments to parameters *file_name* and *list_of_rows,* respectively.
- Step 2
  - A **while** loop was used to construct a loop that continuously displays a menu of choices, then solicits a user for a selection of options available.
  - All lines after the while loop is indented to create a block that groups all logical statement together.
- Step 3
  - Three functions were called to:
    - IO.output_current_tasks_in_list – displays current data in the list
    - IO.output_menu_tasks – displays menu tasks
    - IO.input_menu_choice – obtains menu choice from the user. The choice generated was returned as the return value of the function, then assigned to variable *choice_str* to be used in subsequent operations.

- Step 4
    - *choice_str*, the variable that contains user option choice is evaluated using the if…elif sequence.
        - If *choice_str* equals 1, then Option 1 is chosen to add a new task
            - Functions **IO.input_new_task_and_priority** and **Processor.add_data_to_list** were called to accomplish this task.
            - The **elif** statement is bookended with continue to force the loop to jump back to the beginning and display the menu and asks the user to make the next choice.
        - If *choice_str* equals 2, then Option 2 is chosen to remove an existing task
            - Functions **IO.input_task_to_remove** and **Processor.remove_data_from_list** were called to accomplish this task.
            - The **elif** statement is bookended with continue to force the loop to jump back to the beginning and display the menu and asks the user to make the next choice.
        - If *choice_str* equals 3, then Option 3 is chosen to save data to file
            - Functions **Processor .write_data_to_file** was called to accomplish this task.
            - A confirmation message "Data Saved!" is displayed.
            - The **elif** statement is bookended with continue to force the loop to jump back to the beginning and display the menu and asks the user to make the next choice.
        - If *choice_str* equals 4, then Option 4 is chosen to exit the program
            - A confirmation message "Goodbye!" is displayed.
            - The **elif** statement is bookended with **break** to break the while loop and end the program.

Running the Script

In PyCharm

Figure 16. The original ToDoList.txt

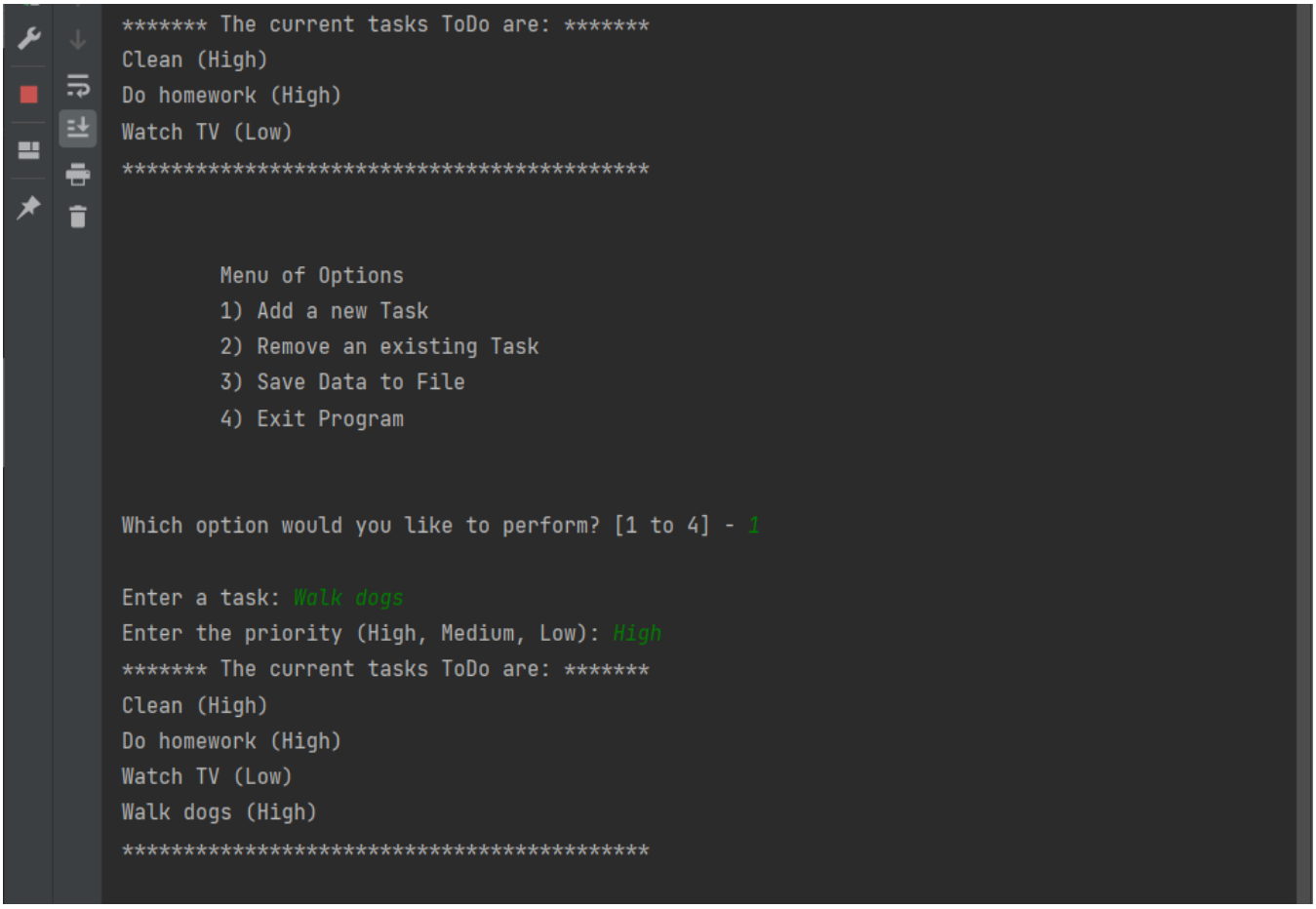

Figure 17. In PyCharm – Option 1

```
        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

Enter a task to remove: Clean
******* The current tasks ToDo are: *******
Do homework (High)
Watch TV (Low)
Walk dogs (High)
*****************************************
```

Figure 18. In PyCharm – Option 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 Saved!
```

Figure 19. In PyCharm – Option 3

ToDoList.txt - Notepad

File Edit Format View Help

```
Do homework,High
Watch TV,Low
Walk dogs,High
```

Figure 20. Updated ToDoList.txt

```
        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!

Process finished with exit code 0
```

Figure 21. In PyCharm – Option 4

- The script has been run successfully in PyCharm.

Running the Script (cont.)

In Command Shell

```
Command Prompt
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Jane>cd C:\Users\Jane\_PythonClass\Assignment06

C:\Users\Jane\_PythonClass\Assignment06>python "C:\Users\Jane\_PythonClass\Assignment06\Assigment06.py"
******* The current tasks ToDo are: *******
Do homework (High)
Watch TV (Low)
Walk dogs (High)
*******************************************


        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: Walk dogs
Enter the priority (High, Medium, Low): High
******* The current tasks ToDo are: *******
Do homework (High)
Watch TV (Low)
Walk dogs (High)
Walk dogs (High)
*******************************************


        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

Enter a task to remove: Walk dogs
******* The current tasks ToDo are: *******
Do homework (High)
Watch TV (Low)
Walk dogs (High)
*******************************************
```

Figure 22. In command shell – Options 1-2

```
Command Prompt

        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 Saved!
******* The current tasks ToDo are: *******
Do homework (High)
Watch TV (Low)
Walk dogs (High)
*******************************************


        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!

C:\Users\Jane\_PythonClass\Assignment06>
```

Figure 23. In command shell – Options 3-4

Summary

The script described above demonstrates several topics and guidelines covered in Module 6. Assignment 06 requirements outlined in Steps 5-10 were successfully carried out and documented in this report. A Python script file is created, using an existing starter script, that manages a “To-do List”. The script uses a printed "menu" to guide the user through various options for the “To-do List”, including displaying current list, adding a new item, removing an existing item, saving data to the original file, and exiting the program. The scope of work is similar to that of Assignment 05, however, the script for Assignment 06 uses functions and classes to organize Data, Processing, and Presentation sections. The "To-do List" file contains two columns of data, "Task" and "Priority." The columns are loaded into a Python dictionary object. Each dictionary object represents one row of data, and these rows are added to a Python list object to create a table of data. This assignment requires updating existing template and code. New tools and concepts introduced in Module 6, such as functions and classes, as well as previously addressed concepts such as list, dictionary, reading and writing data from a file into a list/dictionary will be used in accomplishing this task. Throughout the script, comments were used to describe the functions and intents of the applicable set of codes.

The PyCharm debugging tool was used to troubleshoot and to test out various functions of the tool such as breakpoints and “walk” through the code.

The assignment artifacts were then uploaded to GitHub and Canvas to be submitted for evaluation.