Jessica Trinh
May 19th, 2020
IT FDN 100 A
Assignment 06
https://github.com/trinh-j/IntroToProg-Python-Mod06

Building a Program with Functions and Classes

Introduction --

In this assignment, we will be building on a starter script provided by Professor Randal Root. From it, we will "fill in the blanks" to write a program that allows the user to choose from a menu of options that will allow them to input, save, display, and/or reload data. While this assignment is set up as an exercise for students to learn how to work with another developer's code, it also challenges students to apply newly acquired knowledge of python classes and functions from Module 06. Classes and functions can be intimidating, but hopefully we will gain a better understanding as we go through the assignment.

Basics ---

A **function** is an object that groups variables and statements. A function is initialized by *def* for "define", followed by a function name of your choosing. Here is an example of a simple function, *minus*, that takes in two parameters, number1 and number2, with default assignments of 0, and returns the difference of the two numbers.

```
def minus(number1=0, number2=0):
   return number1 - number2
```

In order for this function to work, we have to call out to it. Below, we have two input functions defined after the *minus* function was created (above); this allows the end-user to provide an input that would be used as arguments in the function callout at the very bottom.

```
number1 = float(input("Enter your first number: " )
number2 = float(input("Enter your second number: " )

minus(number1, number2)
```

Say, what if the user just wants to specify the values upon function callout? Then the arguments they provide would be considered *positional* arguments. Below, since 3 is in place of parameter "number1" and 4 in place of "number2", the output would be -1 since the function defines their operation in its body as "number1 – number2."

```
minus(3,4)
```

A remedy for having more control over the arguments is to simply assign their parameter names. The output of the following example is (positive) 1.

```
minus(number2 = 3, number1 = 4)
```

A **class** is an object that groups together functions. A class is initialized with *class* and the class name is often in snake-casing. In programming, classes are useful because they can be reused in different parts of the program once defined and can even be imported into another program if saved. The body of a class contains functions; but here, the functions are called "class methods."

Below is an example class *math* with class methods *minus* and *plus*.

```
class math:
        def plus(number1=0, number2=0):
                return number1 + number2
        @staticmethod
        def minus(number1=0, number2=0):
                return number1 - number2
```

Like functions, classes have to be called out to be used. There are two ways of doing this;

(1) make a copy of the class, reassign it to a variable, and call out the function:

```
objC = math()
objC.minus(3,4)
```

(2) use a decorator (@staticmethod from second class method in *math* class)—calls out directly to the method:

```
math.minus(3,4)
```

In python 3, the class can be defined with or without parentheses (`class math:` or `class math():` works).

Writing the Script ---

Since we are given a starter script, there is not much to do with regard to creating the script header and defining variables in the Data section. The most we would have to do with the script header is update the change log as necessary.

Here comes the hard part. The sections that we are to build on are Processing, Presentation, and Main Body of Script. Here is an overview of each section we are building on:

Processing section holds all the code that is needed to execute data-entry/appending, data-removal, reading, and writing data; all functions associated with processing is grouped into a *Processor* class.

The **Presentation** section is all the code that displays data to the end-user, and generally requires feedback from the user. All functions associated with user input/output is grouped into the *IO* class

Main Body of Script is the last section in which everything is pieced together. The body of the while loop in this section is mostly composed of class-method callouts.

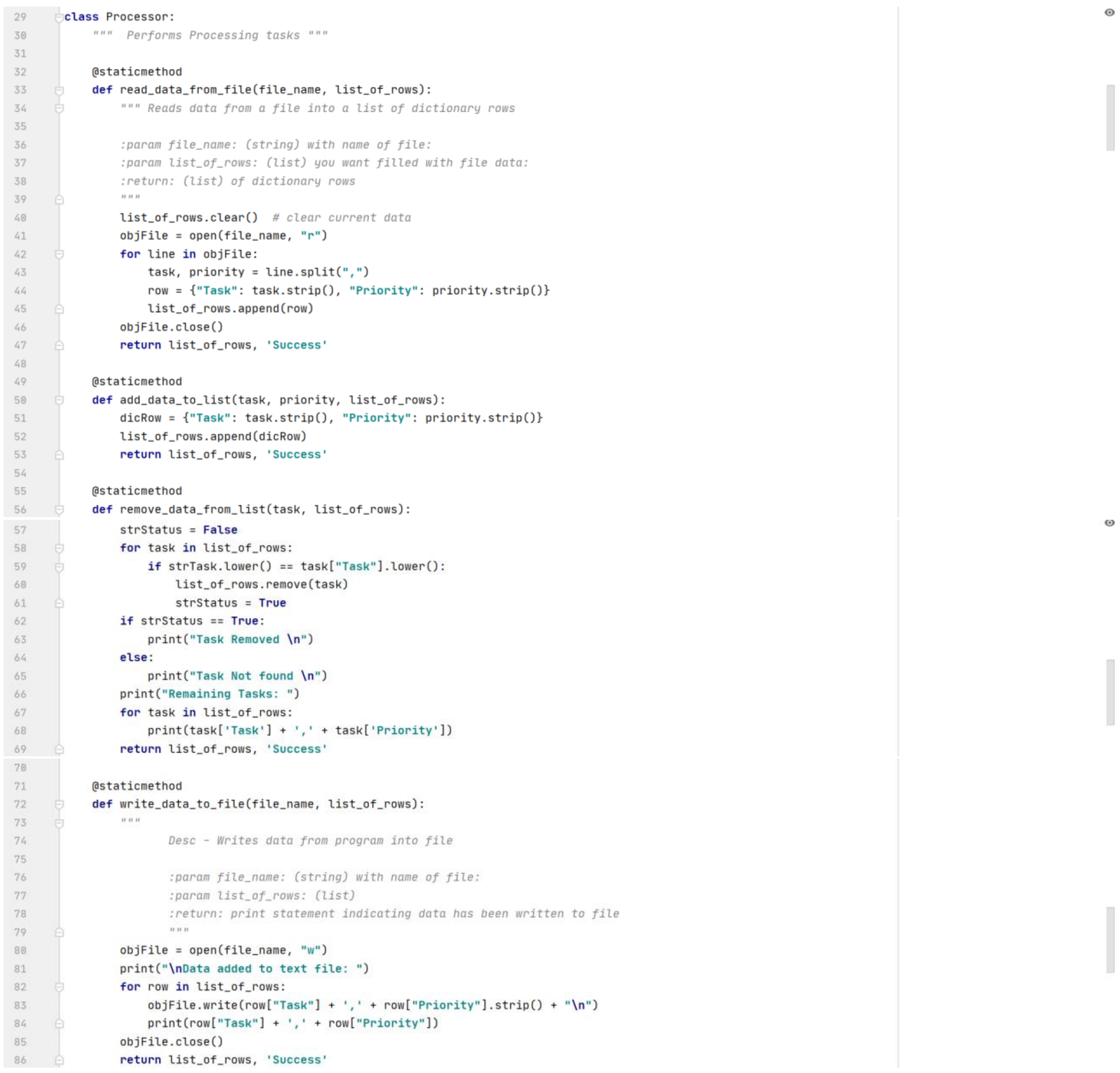
Disclaimer: Though the following demonstration may seem linear, as I am presenting from the beginning to end of my script, the process of building this program was far from it; I had to build one method, check and troubleshoot it in the main body, fix another secondary class-method, go to the main body to troubleshoot again, etc.

Processing

(See Figure 1 for all references made to the script in this section)

As mentioned earlier, this section is one class, *Processor*, composed of four functions or class-methods—*read_data_from_file, add_data_to_list, remove_data_from_list,* and *write_data_to_file*. Each of these class-methods is responsible for processing the data. More specifically, the first class-method reads data from a text file and reformats each row to a dictionary, which is then appended to an empty list. The second class-method reformats the user-input into a dictionary then appends it to the existing (previously empty) list. The third removes a task by iterating through the list until the user's input (in lowercase) matches a task in the list (in lowercase), and removes the matched row. And finally, the last class-method opens, writes data to the text file, and then closes the file, should the end-user want to save any data entered through the program.

On the same line of class initiation, each class requires parameters, for which an associated argument is needed upon callout to make the class-method work. Also, above each class method is a decorator @staticmethod. This will make the function within the class more accessible by allowing the program to directly call it out when it is used later in the script.



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

    @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
        objFile = open(file_name, "r")
        for line in objFile:
            task, priority = line.split(",")
            row = {"Task": task.strip(), "Priority": priority.strip()}
            list_of_rows.append(row)
        objFile.close()
        return list_of_rows, 'Success'

    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):
        dicRow = {"Task": task.strip(), "Priority": priority.strip()}
        list_of_rows.append(dicRow)
        return list_of_rows, 'Success'

    @staticmethod
    def remove_data_from_list(task, list_of_rows):
        strStatus = False
        for task in list_of_rows:
            if strTask.lower() == task["Task"].lower():
                list_of_rows.remove(task)
                strStatus = True
        if strStatus == True:
            print("Task Removed \n")
        else:
            print("Task Not found \n")
        print("Remaining Tasks: ")
        for task in list_of_rows:
            print(task['Task'] + ',' + task['Priority'])
        return list_of_rows, 'Success'

    @staticmethod
    def write_data_to_file(file_name, list_of_rows):
        """
               Desc - Writes data from program into file

               :param file_name: (string) with name of file:
               :param list_of_rows: (list)
               :return: print statement indicating data has been written to file
               """
        objFile = open(file_name, "w")
        print("\nData added to text file: ")
        for row in list_of_rows:
            objFile.write(row["Task"] + ',' + row["Priority"].strip() + "\n")
            print(row["Task"] + ',' + row["Priority"])
        objFile.close()
        return list_of_rows, 'Success'
```

Figure 1. Processing Section of Assignment06.py script.

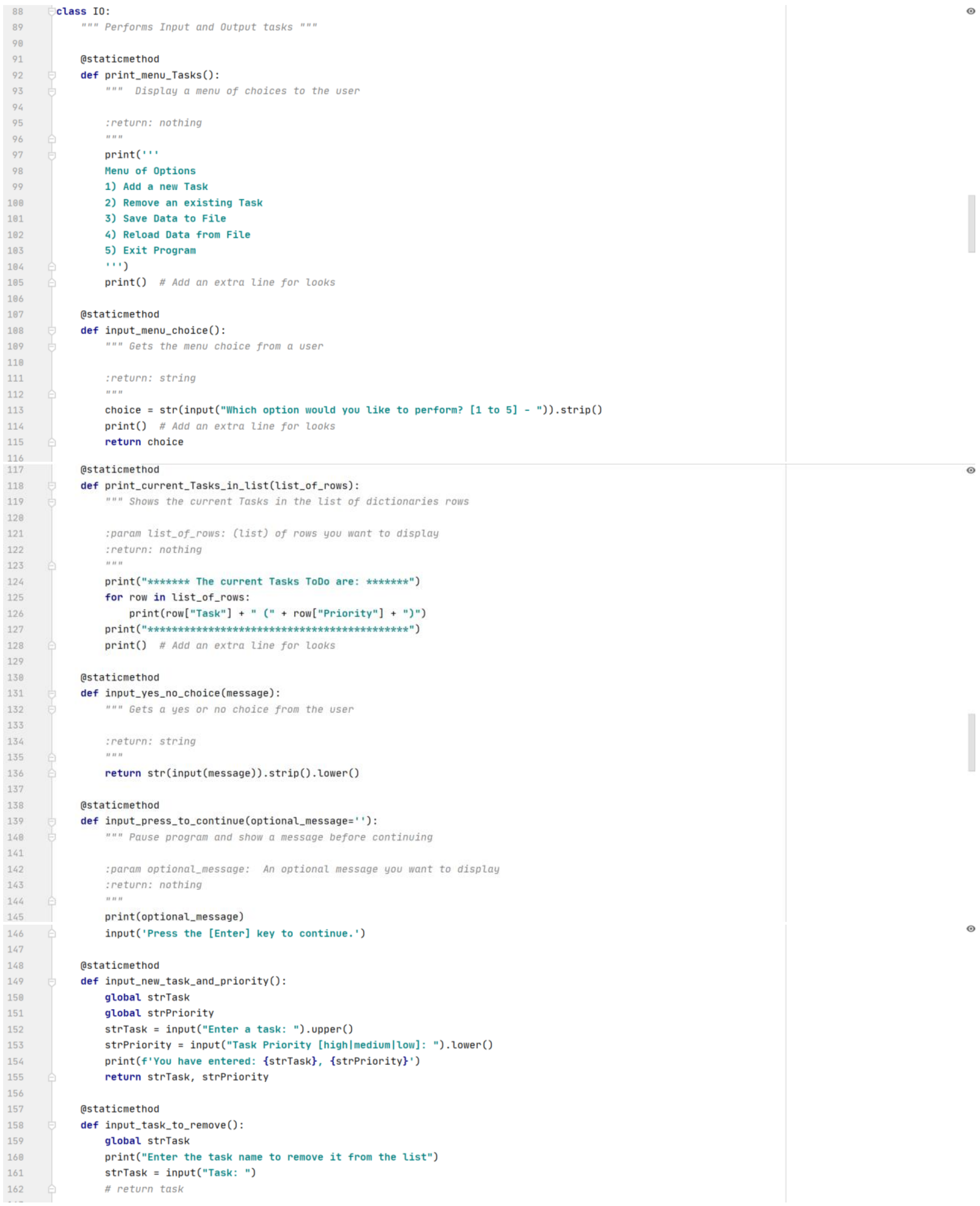
Presentation

(See Figure 2 for all references made to the script in this section, unless noted otherwise.)

This section is also entirely made up of one class, *IO* (for Input Output), which is made of 7 functions. Similar to class *Processor* in the Processing section (Figure 1), each class-method has a decorator above it to make class-method callout easier later in the script. In this section, we see all functions associated with displaying data to the user. For example, in lines 92-105 (Figure 2), we see the Menu of Options being defined as a function, *print_menu_Tasks()*. Since this function is just a simple print function, it needs no parameters/arguments. However, as we have seen before with the functions in the processing section, the parameters vary with each function, and is defined by the developer. In this starter script, Professor root pre-defined the parameters and function names.

In Module 06, we discussed the differences between global and local variables. Global variables exist outside a class or function, and local variables are variables that only exist within the class or function where it is created. In order to access local variables outside the class or function, I used the *global* function (lines 150 -151, Figure 2). Initially, strTask and strPriority were defined as an empty string in the Data section of the script. However, when using it within a class-method, values associated with it were stored in its “local” memory, and I was not able to access the associated end-user inputs outside of the class-method. After reprocessing strTask and strPriority as global variables within my class-method, I was able to access/return the values and use them in another class-method.

Working with each class-method can be daunting, especially if there is a class-method that relies on another class method. Again, there was a lot of back-and-forth coding/troubleshooting that was required for me to get the program to work as it does now.



```
class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def print_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) Reload Data from File
        5) Exit Program
        ''')
        print()  # Add an extra line for looks

    @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 5] - ")).strip()
        print()  # Add an extra line for looks
        return choice

    @staticmethod
    def print_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

    @staticmethod
    def input_yes_no_choice(message):
        """ Gets a yes or no choice from the user

        :return: string
        """
        return str(input(message)).strip().lower()

    @staticmethod
    def input_press_to_continue(optional_message=''):
        """ Pause program and show a message before continuing

        :param optional_message:  An optional message you want to display
        :return: nothing
        """
        print(optional_message)
        input('Press the [Enter] key to continue.')

    @staticmethod
    def input_new_task_and_priority():
        global strTask
        global strPriority
        strTask = input("Enter a task: ").upper()
        strPriority = input("Task Priority [high|medium|low]: ").lower()
        print(f'You have entered: {strTask}, {strPriority}')
        return strTask, strPriority

    @staticmethod
    def input_task_to_remove():
        global strTask
        print("Enter the task name to remove it from the list")
        strTask = input("Task: ")
        # return task
```

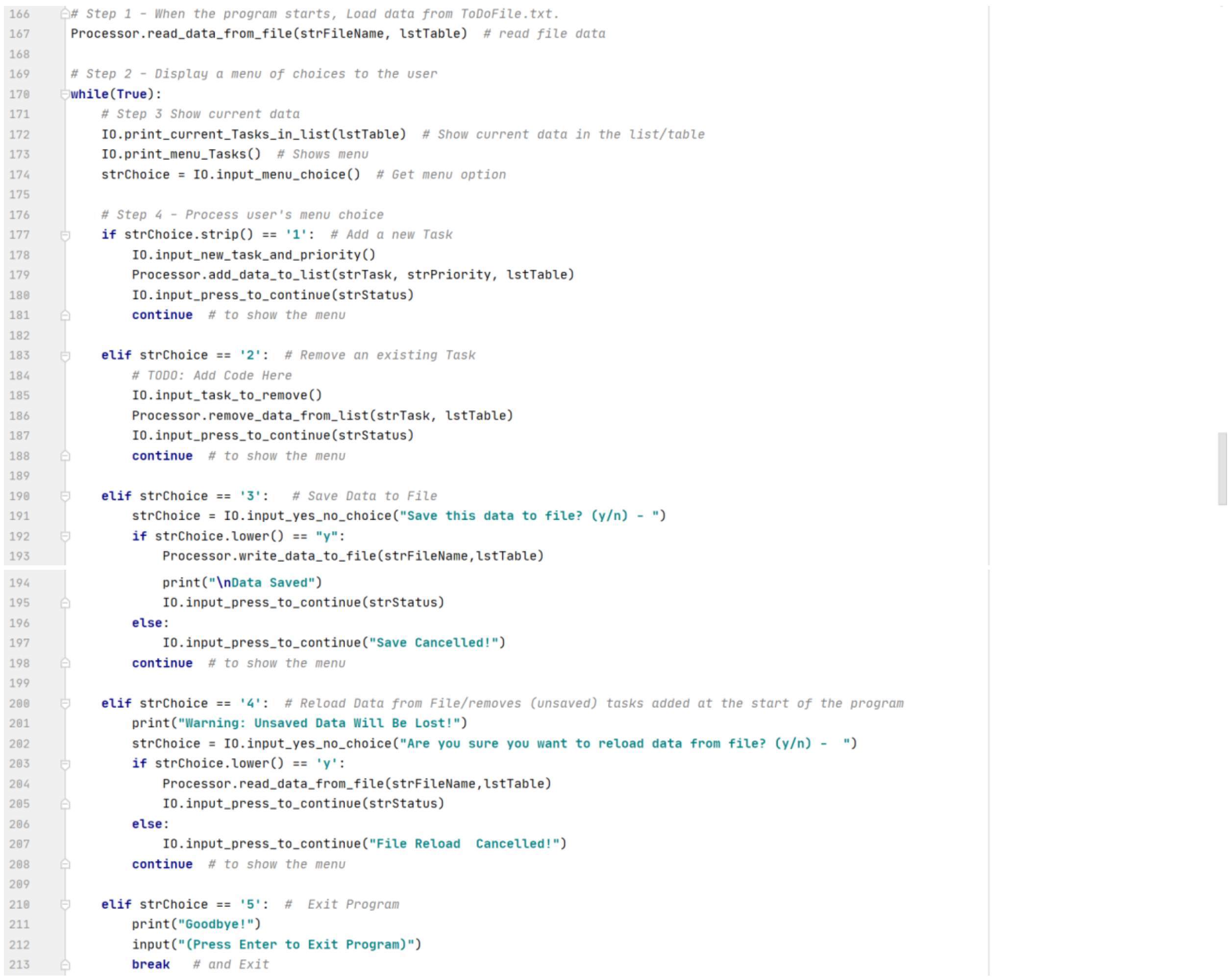
Figure 2. Presentation Section of Assignment06.py script. Class IO with 7 input-output-associated class-methods in the body.

Main Body of Script

(See Figure 3 for all references made to the script in this section, unless noted otherwise.)

Finally, the Main Body. In this section, we piece together a program using primarily class-methods previously defined in the Processing (Figure 1) and the Presentation (Figure 2) sections of the script. Since we "prefaced" each class-method with a decorator, we can call out to it in the form *class.[class-method](arguments) (See "Basics" section above).*

The main body (Figure 3) uses a while-loop to filter through different menu options based on end-user input.



```
166 # Step 1 - When the program starts, Load data from ToDoFile.txt.
167 Processor.read_data_from_file(strFileName, lstTable)  # read file data
168
169 # Step 2 - Display a menu of choices to the user
170 while(True):
171     # Step 3 Show current data
172     IO.print_current_Tasks_in_list(lstTable)  # Show current data in the list/table
173     IO.print_menu_Tasks()  # Shows menu
174     strChoice = IO.input_menu_choice()  # Get menu option
175
176     # Step 4 - Process user's menu choice
177     if strChoice.strip() == '1':  # Add a new Task
178         IO.input_new_task_and_priority()
179         Processor.add_data_to_list(strTask, strPriority, lstTable)
180         IO.input_press_to_continue(strStatus)
181         continue  # to show the menu
182
183     elif strChoice == '2':  # Remove an existing Task
184         # TODO: Add Code Here
185         IO.input_task_to_remove()
186         Processor.remove_data_from_list(strTask, lstTable)
187         IO.input_press_to_continue(strStatus)
188         continue  # to show the menu
189
190     elif strChoice == '3':   # Save Data to File
191         strChoice = IO.input_yes_no_choice("Save this data to file? (y/n) - ")
192         if strChoice.lower() == "y":
193             Processor.write_data_to_file(strFileName,lstTable)
194             print("\nData Saved")
195             IO.input_press_to_continue(strStatus)
196         else:
197             IO.input_press_to_continue("Save Cancelled!")
198         continue  # to show the menu
199
200     elif strChoice == '4':  # Reload Data from File/removes (unsaved) tasks added at the start of the program
201         print("Warning: Unsaved Data Will Be Lost!")
202         strChoice = IO.input_yes_no_choice("Are you sure you want to reload data from file? (y/n) -  ")
203         if strChoice.lower() == 'y':
204             Processor.read_data_from_file(strFileName,lstTable)
205             IO.input_press_to_continue(strStatus)
206         else:
207             IO.input_press_to_continue("File Reload  Cancelled!")
208         continue  # to show the menu
209
210     elif strChoice == '5':  #  Exit Program
211         print("Goodbye!")
212         input("(Press Enter to Exit Program)")
213         break   # and Exit
```

Figure 3. Main Body of Script Section of Assignment06.py. The body of each if-elif section using class-method callouts.

Looking at this script, it can be confusing understanding how it works. Let's do a quick example walkthrough:

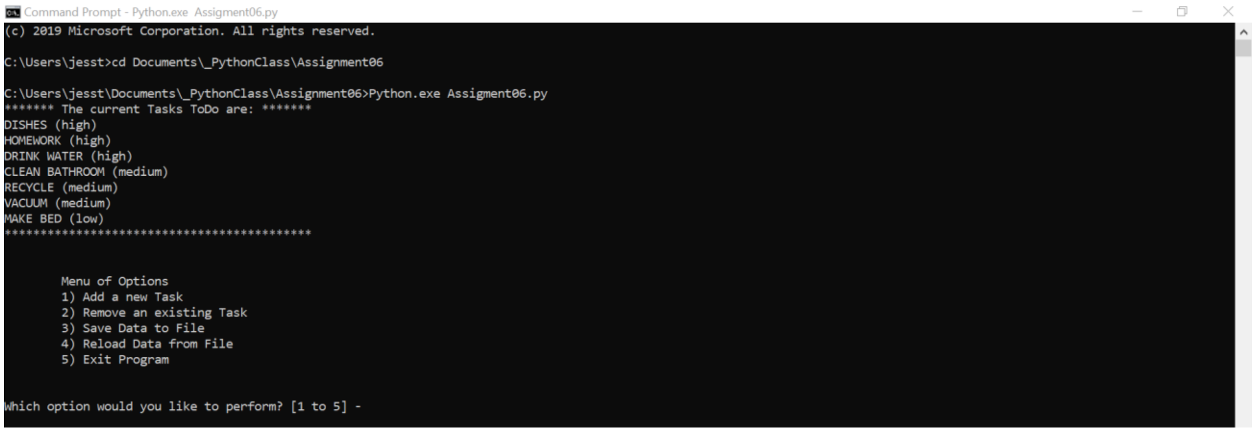
Before the progam starts, it reads data from the text file (line 166, Figure 3) using the class method mentioned in lines 32-47 (Figure 1). Once the program runs through these lines, it jumps back to the main body and continues to the while-loop on line 170 (Figure 3). Since nothing outside the while loop is detected as False, we continue to the body of the while-loop. When the program runs line 172, it jumps up to where the class method is defined, executes lines 117-128 (Figure 2), then jumps back down to process line 173 (Figure 3). When line 173 runs, the program jumps up to the *IO* class again to find the class-method in lines 91-105 (Figure 2), runs those lines, then jumps down to the main body to process line 174 (Figure 3). The program runs line 174, jumps to the class-method in lines 107-115 (Figure 2) to ask for the user input, and returns the input and stores it in variable strChoice. The program then uses the end-user's response to filter through the following if-elif-else (IEE) conditionals in lines 177-213 and if the statemet is true, a series of class-methods will be executed to perform the tasks associated with menu options.

Let's say the end-user's input is "1"; the program will filter through the (IEE) conditionals, for which the input holds true for the if-statement on line 177. Similar to the program ran at the very beginning of the while-loop (lines 172-174, Figure 3), lines 178-180 will be ran and the program will jump to and from the Main Body to the lines where the class-method is defined. After running through those class-methods, which in this case allow the user to input and append data to a list, the program will continue and ask the user if they would like to perform any other operations from the menu of options.

The rest of the program in the while loop, with different strChoice inputs from the user –"2", "3", "4", and "5"—operate similar to the explanation above for strChoice = "1"; the program will weave back and forth from the body of the statement/class-method callout, to where the class-method is defined, and execute the code there before jumping back down to the main body.

Output --

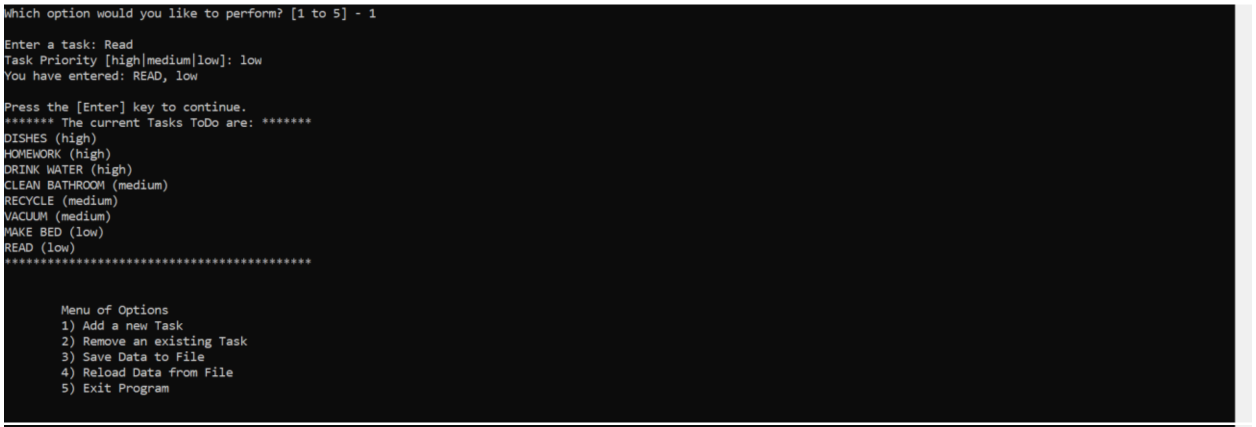
Working with the script in the PyCharm IDE, I knew that my program worked in the console. To double-check program functionality, I accessed my script through the command line, which immediately displayed the current tasks in the text file, and a menu of options that the end-user is prompted to choose from (Figure 4).



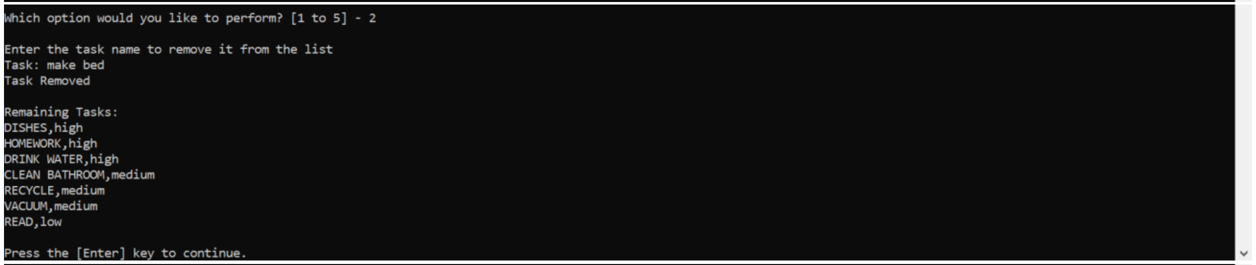
```
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\jesst>cd Documents\_PythonClass\Assignment06

C:\Users\jesst\Documents\_PythonClass\Assignment06>Python.exe Assigment06.py
******* The current Tasks ToDo are: *******
DISHES (high)
HOMEWORK (high)
DRINK WATER (high)
CLEAN BATHROOM (medium)
RECYCLE (medium)
VACUUM (medium)
MAKE BED (low)
*******************************************


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


Which option would you like to perform? [1 to 5] -
```

Figure 4. (Above) Script ran from the command line.

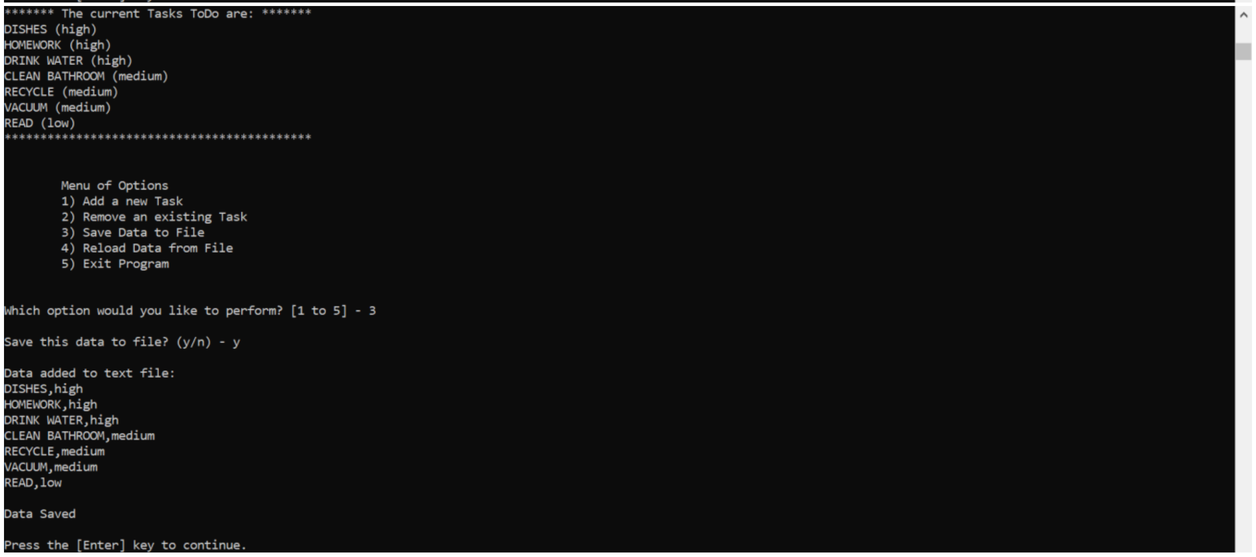
As the user provides an input from the menu of options, the most updated data is displayed to the user (Figure 5). (Unfortunately, the formatting of these images aren't aligned but the following 5 images belong to a single figure-Figure 5).



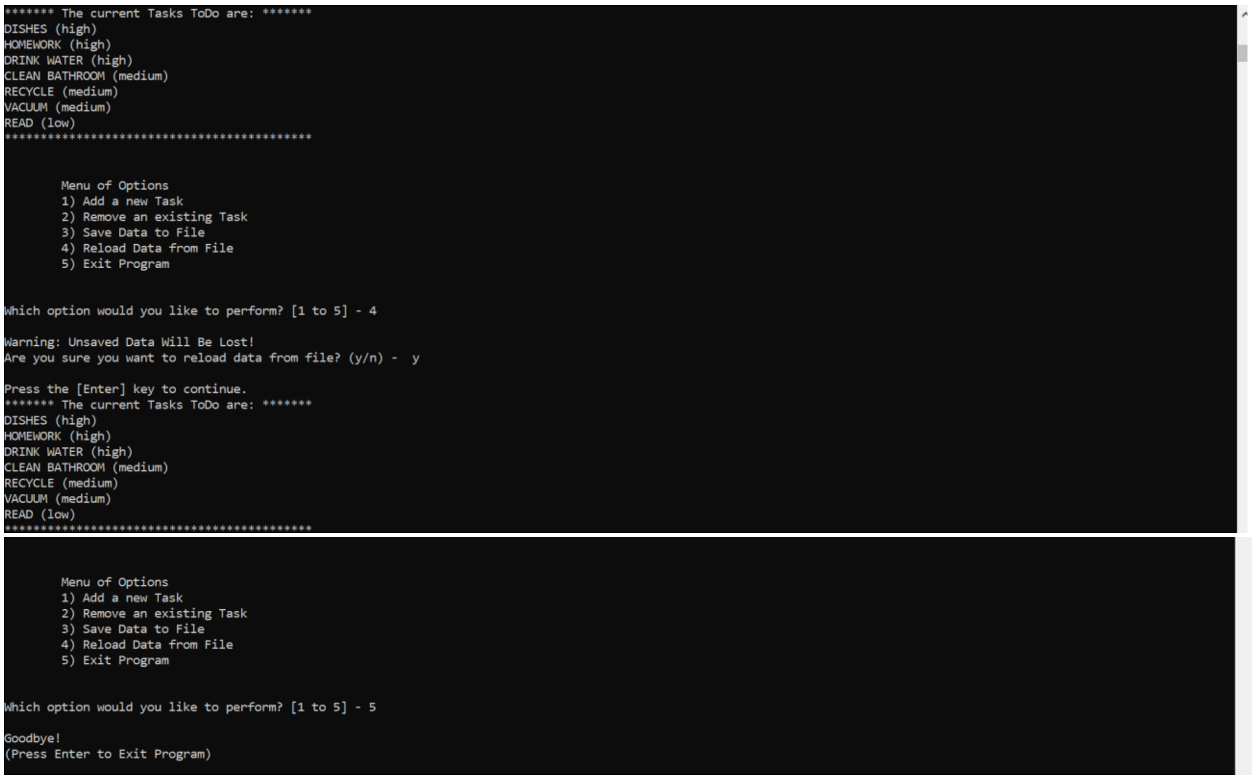
```
Which option would you like to perform? [1 to 5] - 1

Enter a task: Read
Task Priority [high|medium|low]: low
You have entered: READ, low

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
DISHES (high)
HOMEWORK (high)
DRINK WATER (high)
CLEAN BATHROOM (medium)
RECYCLE (medium)
VACUUM (medium)
MAKE BED (low)
READ (low)
*******************************************


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



```
Which option would you like to perform? [1 to 5] - 2

Enter the task name to remove it from the list
Task: make bed
Task Removed

Remaining Tasks:
DISHES,high
HOMEWORK,high
DRINK WATER,high
CLEAN BATHROOM,medium
RECYCLE,medium
VACUUM,medium
READ,low

Press the [Enter] key to continue.
```



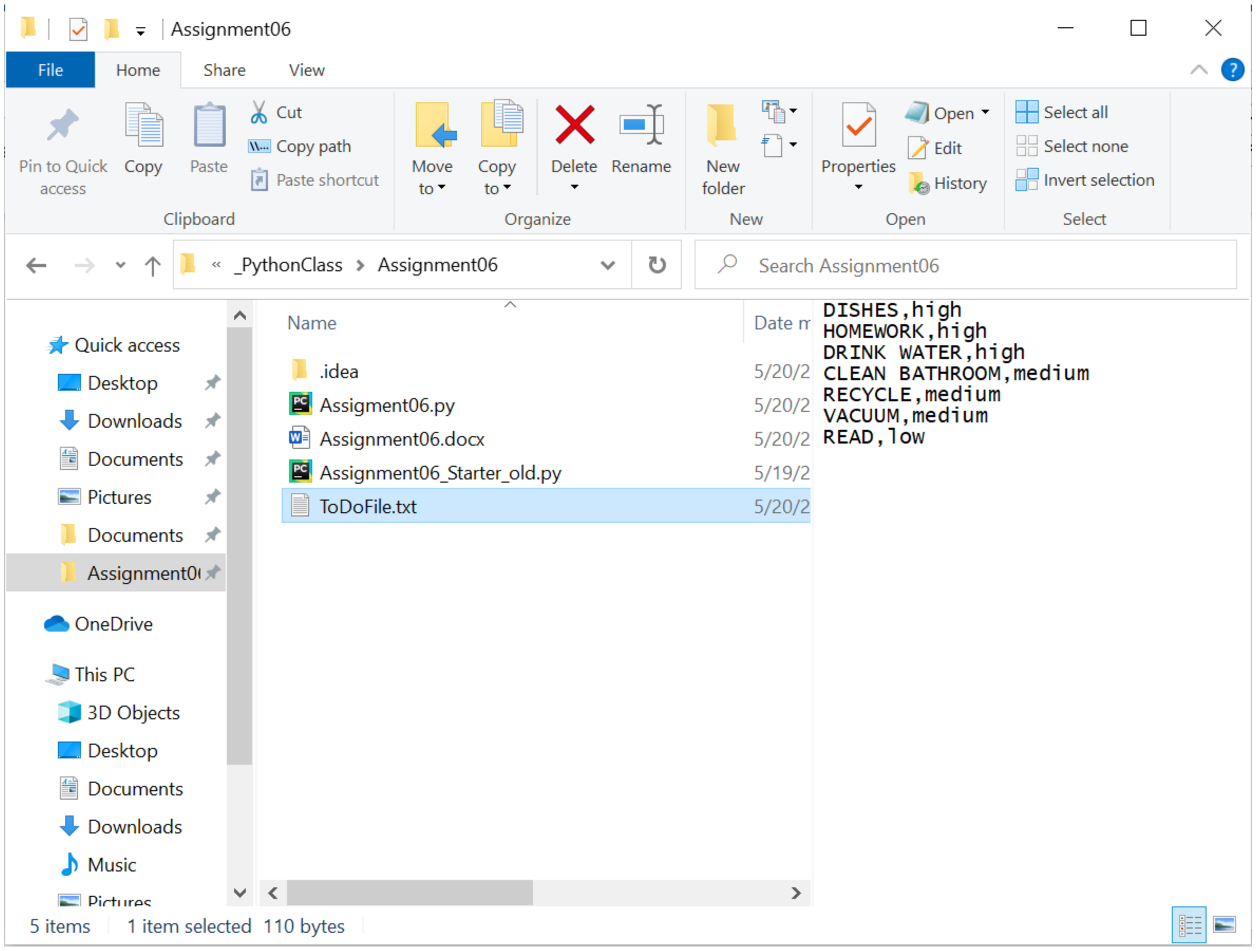
```
******* The current Tasks ToDo are: *******
DISHES (high)
HOMEWORK (high)
DRINK WATER (high)
CLEAN BATHROOM (medium)
RECYCLE (medium)
VACUUM (medium)
READ (low)
*******************************************


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


Which option would you like to perform? [1 to 5] - 3

Save this data to file? (y/n) - y

Data added to text file:
DISHES,high
HOMEWORK,high
DRINK WATER,high
CLEAN BATHROOM,medium
RECYCLE,medium
VACUUM,medium
READ,low

Data Saved

Press the [Enter] key to continue.
```



```
******* The current Tasks ToDo are: *******
DISHES (high)
HOMEWORK (high)
DRINK WATER (high)
CLEAN BATHROOM (medium)
RECYCLE (medium)
VACUUM (medium)
READ (low)
*******************************************


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


Which option would you like to perform? [1 to 5] - 4

Warning: Unsaved Data Will Be Lost!
Are you sure you want to reload data from file? (y/n) -  y

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
DISHES (high)
HOMEWORK (high)
DRINK WATER (high)
CLEAN BATHROOM (medium)
RECYCLE (medium)
VACUUM (medium)
READ (low)
*******************************************


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


Which option would you like to perform? [1 to 5] - 5

Goodbye!
(Press Enter to Exit Program)
```

Figure 5. Program in command line with all outputs for each menu option.

To make sure our data is saved to the text file, we can navigate to the appropriate directory using the file explorer. Here (Figure 6), you can see that the file explorer contains the text file, and in the preview section to the right, the list has contains the updated list resulting from the execution in the command line above (Figure 5).



```
DISHES,high
HOMEWORK,high
DRINK WATER,high
CLEAN BATHROOM,medium
RECYCLE,medium
VACUUM,medium
READ,low
```

Figure 6. Use File Explorer to navigate to project directory; ToDoFile.txt found with updated list in preview panel.

If you don’t have a preview panel, you can set it up by going to the View tab, and selecting “Preview Pane.” Otherwise you can open the text file to see any changes made.

Summary/Discussion ---

In this assignment, we learned how to use classes and functions to organize a program. This was definitely a tough assignment, as I spent quite a while just reading through the script to try to figure it out. Still, after understanding what the program was trying to do and starting to add some code, I quickly found myself scrolling back and forth to understand the order of code being processed, and mitigating frequently arising issues by commenting out specific lines of code, and using the debugging tool. For data removal, in Assignment05, I had trouble displaying the correct statement to the user; if data was removed, “Task not found” would sometimes be printed instead of “Task removed.” For this assignment, I reviewed the answer key to Assignment05 and found that to do what I want it to do—print “removed” for data removal and “not found” if data isn’t in the list—I had to use a Boolean statement with my if-conditional statement. I incorporated this into my latest code. A concept I’ve yet to fully understand is the reassignment of a local variable as a global variable. Though this project was challenging, I enjoyed solving it.