Erica Peterson

August 15th, 2022

Foundations of Programming: Python

Assignment 06

https://github.com/ericapet/ITFDN110B-Mod06

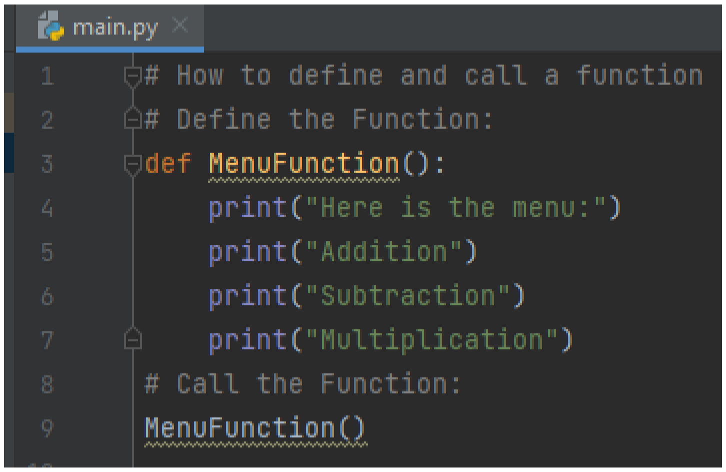
Functions

Introduction

In this paper, I will go over functions and classes and how to use them to simplify and organize your code. I will essentially be using the same program from last week, assignment 05, but adding in functions and classes to make it more organized.

Functions

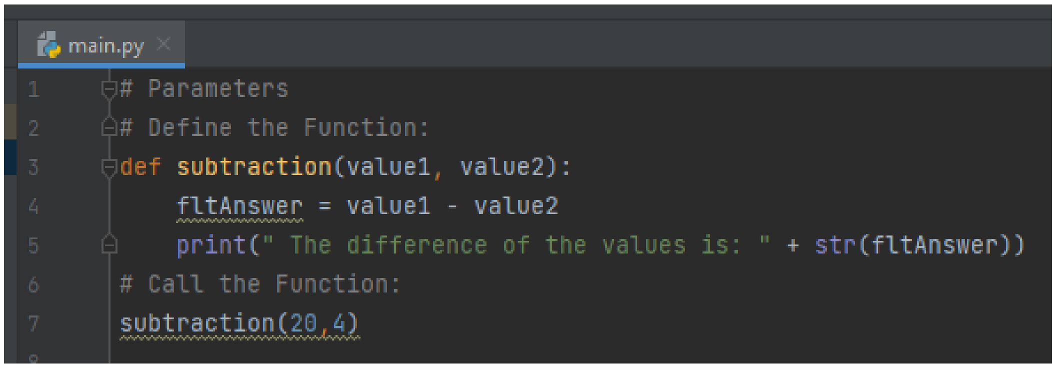
Functions are a way to group one or more statements to run under one name. You must first define the function before you can call the function to run. Calling the function runs all the statements in the function definition. To define a function, you use def and then the name of the function. An example of a simple function is seen in Figure 1.



```python
# How to define and call a function
# Define the Function:
def MenuFunction():
    print("Here is the menu:")
    print("Addition")
    print("Subtraction")
    print("Multiplication")
# Call the Function:
MenuFunction()
```

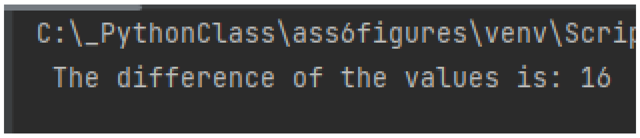
Figure 1. How to define and call a function. Notice that there are multiple print statements in the function.

Functions can also have parameters, which allow you to pass values into the function for processing. They are a variable placeholder for the actual values the function will need while running. These parameters can also be called arguments. There is also no limit on how many parameters you can include. Typically, you define parameter names without a prefix. An example of a function with parameters can be found in Figure 2 on the following page.



main.py

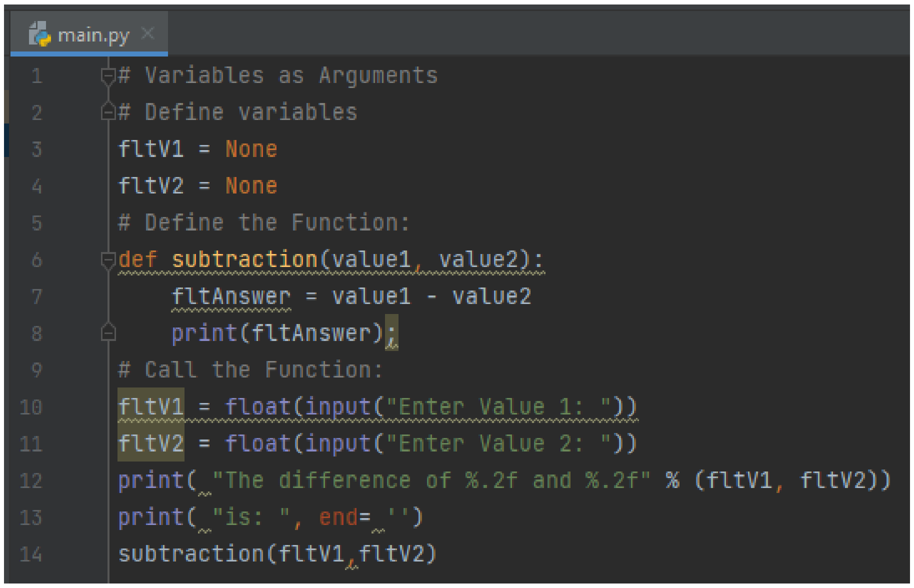
```python
# Parameters
# Define the Function:
def subtraction(value1, value2):
    fltAnswer = value1 - value2
    print(" The difference of the values is: " + str(fltAnswer))
# Call the Function:
subtraction(20,4)
```



```
C:\_PythonClass\ass6figures\venv\Scrip
 The difference of the values is: 16
```

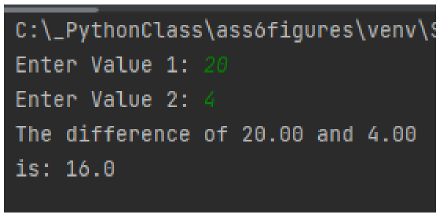
Figure 2. An example of a function that performs subtraction of two parameters (value1, value2). The bottom image shows the output when the function is run.

You can also use variables as arguments in your function, this is useful when you want your script to access these values repeatedly. You do this by defining the variables outside of the function, using the none data type. An example of using variables as arguments can be seen in Figure 3, below.



main.py

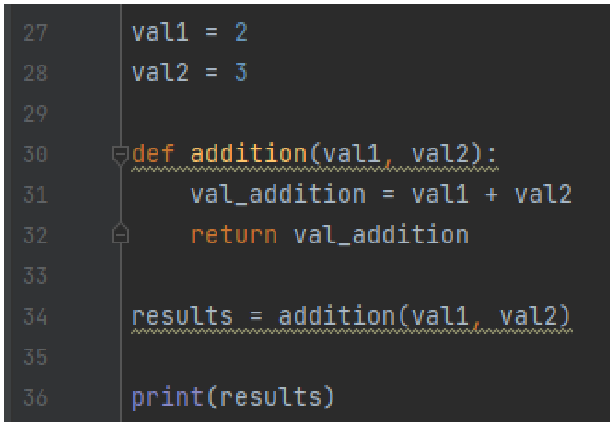
```python
# Variables as Arguments
# Define variables
fltV1 = None
fltV2 = None
# Define the Function:
def subtraction(value1, value2):
    fltAnswer = value1 - value2
    print(fltAnswer);
# Call the Function:
fltV1 = float(input("Enter Value 1: "))
fltV2 = float(input("Enter Value 2: "))
print( "The difference of %.2f and %.2f" % (fltV1, fltV2))
print( "is: ", end= '')
subtraction(fltV1,fltV2)
```



```
C:\_PythonClass\ass6figures\venv\S
Enter Value 1: 20
Enter Value 2: 4
The difference of 20.00 and 4.00
is: 16.0
```

Figure 3. Using Variables as arguments; script shown in the top image, output in bottom image.

Functions can also return one or more values and it can be any python object such as int, float, list, tuple, or dictionary. This return must be used inside the function. The return will return the value of the expression following the return keyword. An example using return is shown in Figure 4.

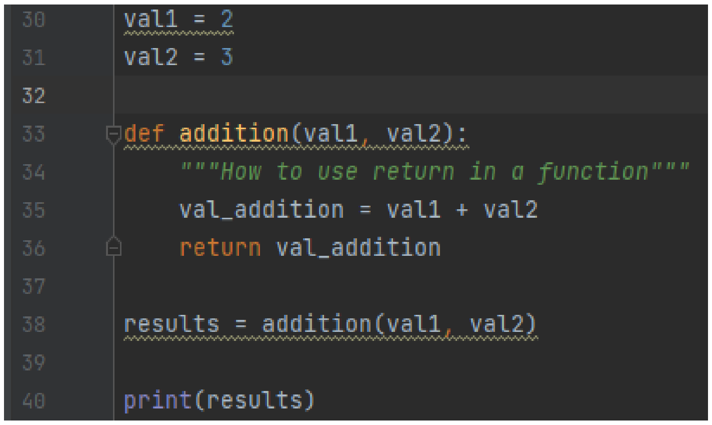


```
27  val1 = 2
28  val2 = 3
29
30  def addition(val1, val2):
31      val_addition = val1 + val2
32      return val_addition
33
34  results = addition(val1, val2)
35
36  print(results)
```

Figure 4. How to use return in a function. Notice in line 34 you must assign a variable name to your return value.

Variables can be global or local, depending on where they are in the script. Variables declared inside a function are considered local and cannot be used outside of that function. Variables declared outside of a function are considered global and can be used anywhere in the script. In python, you usually would use local variables inside your function and not a global variable from outside it, if you do use a global variable within a function, it is best practice to use the keyword global.

Another part of creating your own function is to add a document header to describe what your function does. This is known as a docstring in python. To add a function document header, you use triple quotes “““document header”””. An example of this can be found in Figure 5.

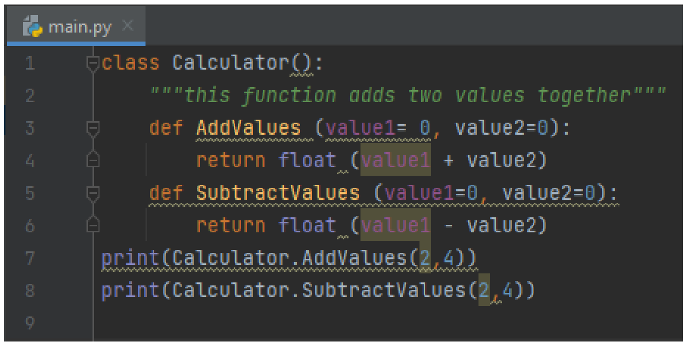


```
30  val1 = 2
31  val2 = 3
32
33  def addition(val1, val2):
34      """How to use return in a function"""
35      val_addition = val1 + val2
36      return val_addition
37
38  results = addition(val1, val2)
39
40  print(results)
```

Figure 5. Adding a function document header. The docstring can be seen in line 34.

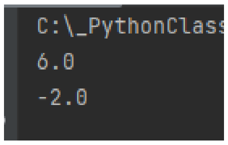
Classes

Classes are used to group functions, constants, and variables. Classes help with code organization. Class definition being with the class keyword. Like with definitions, the first string inside of a class should be a docstring, which has a brief description of the class. An example of a class is seen below in Figure 6.



main.py

```python
class Calculator():
    """this function adds two values together"""
    def AddValues (value1= 0, value2=0):
        return float (value1 + value2)
    def SubtractValues (value1=0, value2=0):
        return float (value1 - value2)
print(Calculator.AddValues(2,4))
print(Calculator.SubtractValues(2,4))
```

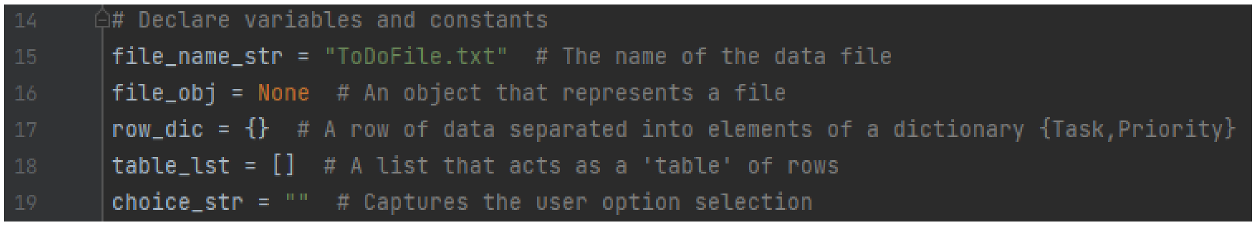


```
C:\_PythonClass
6.0
-2.0
```

Figure 6. An example of a class, notice the docstring in line 2. The bottom image shows the output, correctly adding and subtracting 2 and 4 to 6 and -2.

How To Write a Script for a To Do List, Using Functions

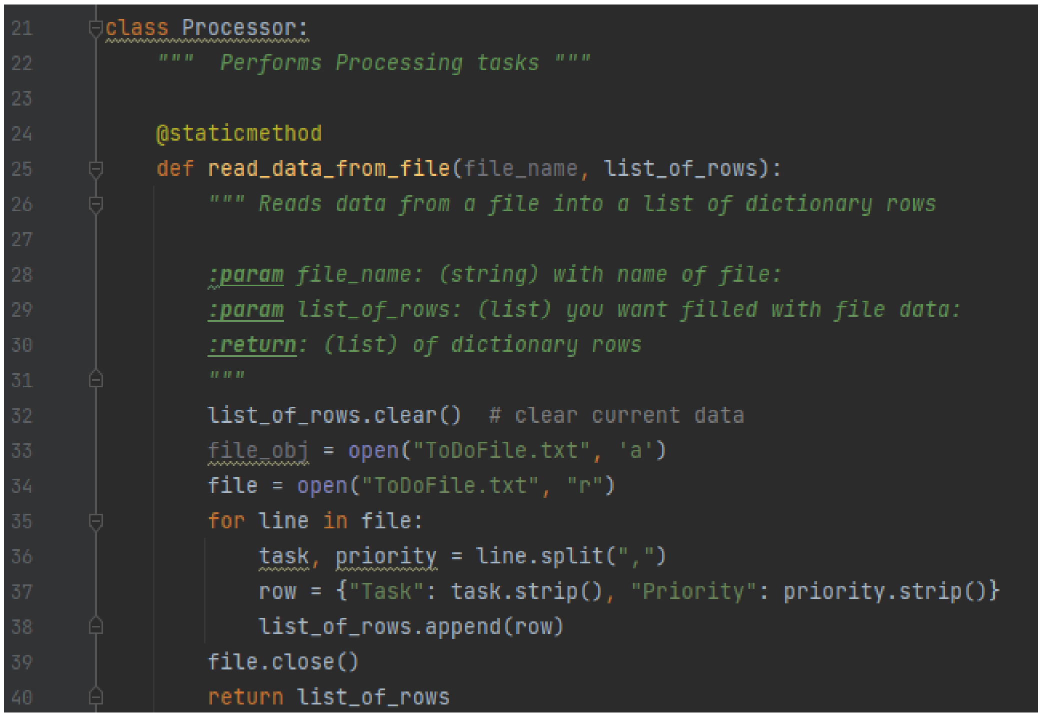
Now that we know a bit more about functions and classes, we can edit a script for a to do list, adding in functions to make the script more organized. This script is very similar to Assignment 05, with some of the variable names changed. The first step is to declare our variables and constants at the beginning of the code. This makes it easier for other programmers to read your code and edit it. My code for declaring variables can be found in Figure 7, below.



```python
# Declare variables and constants
file_name_str = "ToDoFile.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,Priority}
table_lst = []  # A list that acts as a 'table' of rows
choice_str = ""  # Captures the user option selection
```

Figure 7. Code block for declaring variables and constants, notice that each variable/constant has a comment description.

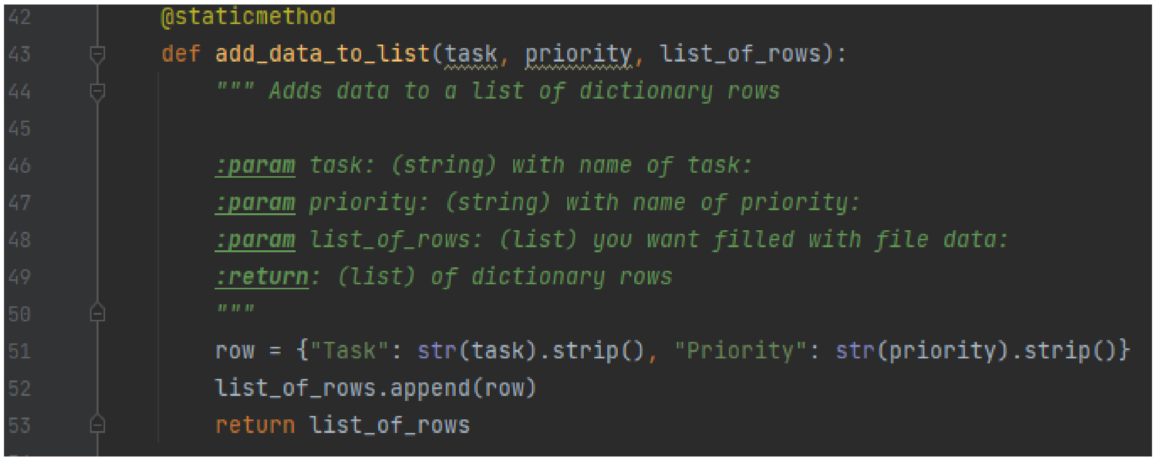
The next step is to create our class for processing, filled with functions to do each part of the to do list. The first function is to read data from the file. How to make this function is seen on the next page, in Figure 8.



```
21  class Processor:
22      """  Performs Processing tasks """
23
24      @staticmethod
25      def read_data_from_file(file_name, list_of_rows):
26          """ Reads data from a file into a list of dictionary rows
27
28          :param file_name: (string) with name of file:
29          :param list_of_rows: (list) you want filled with file data:
30          :return: (list) of dictionary rows
31          """
32          list_of_rows.clear()  # clear current data
33          file_obj = open("ToDoFile.txt", 'a')
34          file = open("ToDoFile.txt", "r")
35          for line in file:
36              task, priority = line.split(",")
37              row = {"Task": task.strip(), "Priority": priority.strip()}
38              list_of_rows.append(row)
39          file.close()
40          return list_of_rows
```

Figure 8. How to create a class for processing and how to make the first function in the class, to read data from the file. The class, called processor, is in line 21. The first function, read_data_from_file, is seen in rows 25-40. Lines 32-40 are essentially copied from Assignment 05 but the variable names were changed to match the existing code provided by Randal Root.

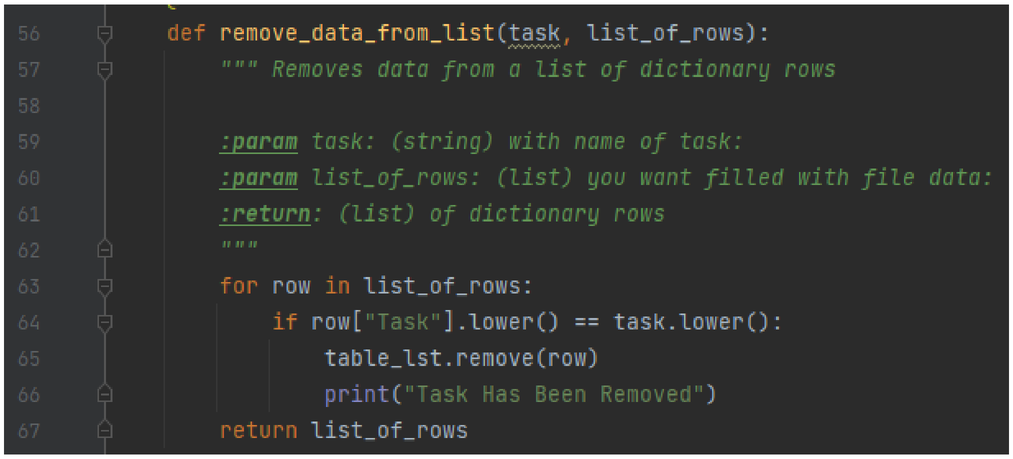
The next function adds data to the list. The code in this block is also very similar to Assignment 05. The add_data_to_list function can be found in Figure 9, below. To add data we use the .append function.



```
    @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 9. How to create a function that adds data to a list of dictionary rows.

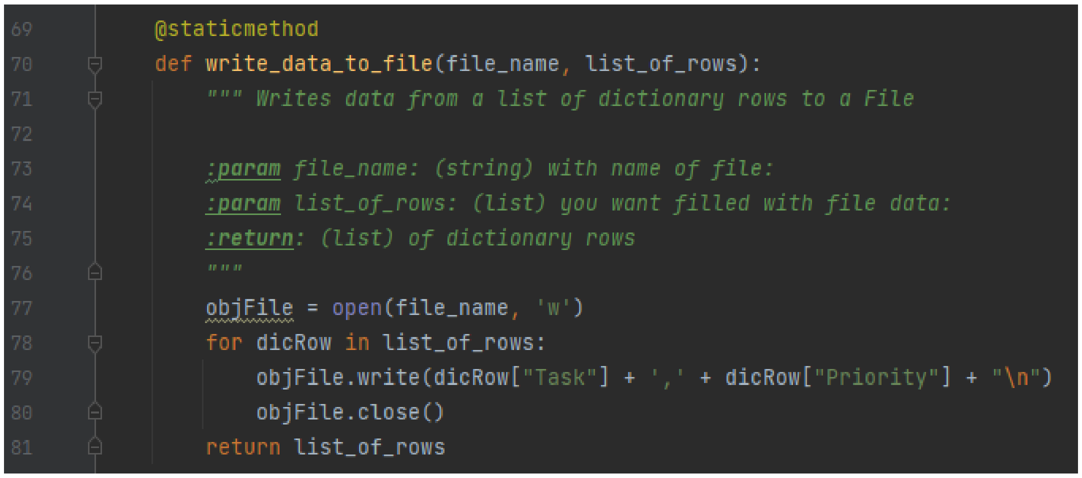
Next, we must write a function to remove data from the list. This is once again, very similar to Assignment 05. Similar to adding data, we use the .remove function to remove data. The code also must be able to find the matching task the user asked to remove, this can be seen in line 64 in Figure 10. The entire remove_data_from_list function is depicted in Figure 10, below.



```
56      def remove_data_from_list(task, list_of_rows):
57          """ Removes data from a list of dictionary rows
58  
59          :param task: (string) with name of task:
60          :param list_of_rows: (list) you want filled with file data:
61          :return: (list) of dictionary rows
62          """
63          for row in list_of_rows:
64              if row["Task"].lower() == task.lower():
65                  table_lst.remove(row)
66                  print("Task Has Been Removed")
67          return list_of_rows
```

Figure 10. How to write a function to remove data from the list of dictionary rows.

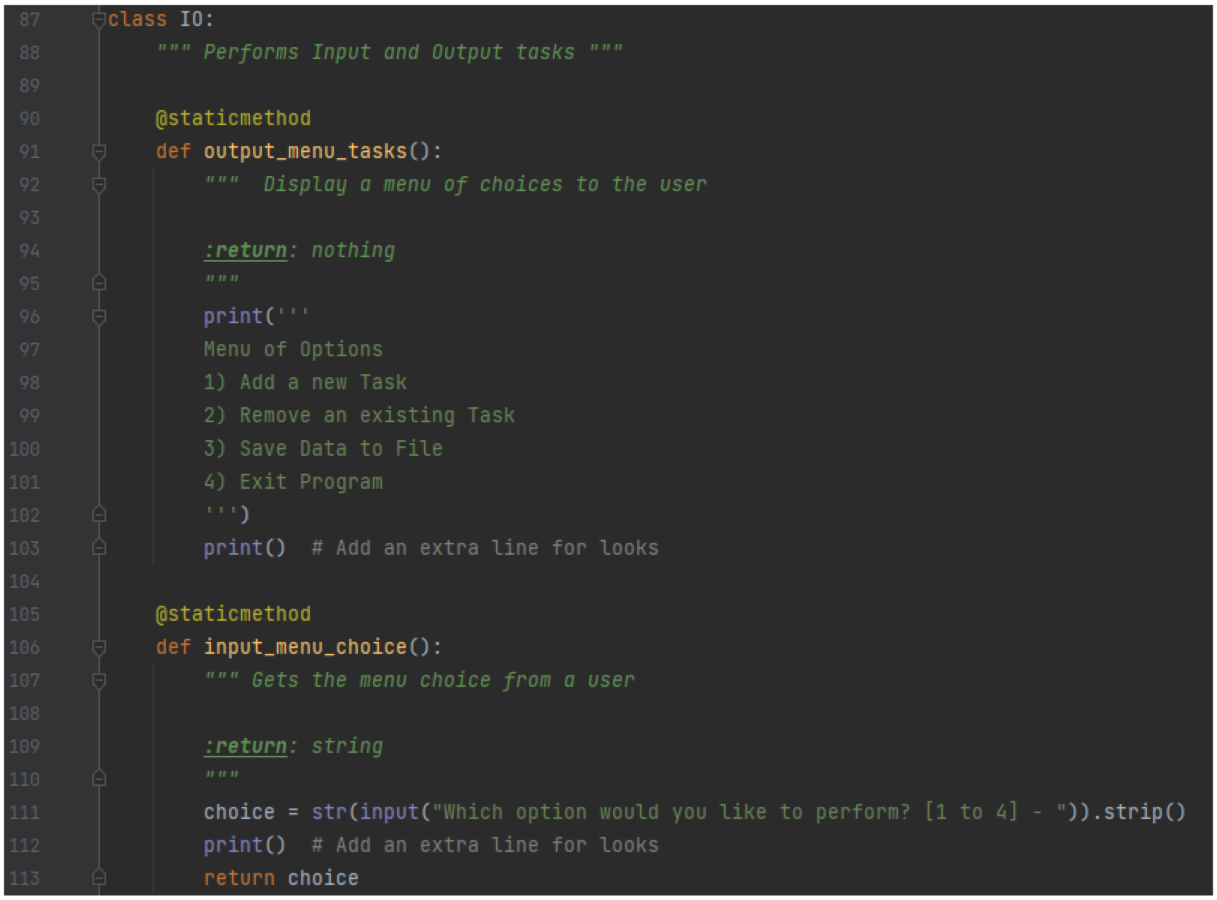
The final function we need to write will write data to the file. For this, we will need to open our file in write mode and then write the data to the file. The code block for accomplishing this can be found in Figure 11, below.



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

Figure 11. How to write data to a file, in a function.

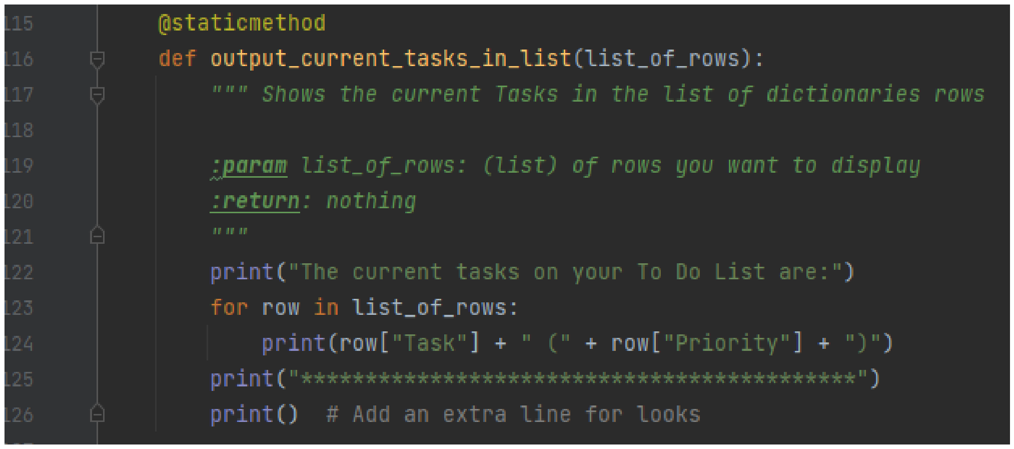
Now we will create a new class for the input and output parts of this script. This class will be called IO. The first function in this class is to display a menu to the user so that they can input what they want to do. The second function is to collect user input on which menu choice they select. The IO class, output_menu_tasks and input_menu_choice functions can be seen in Figure 12, on the following page.



```
87  class IO:
88      """ Performs Input and Output tasks """
89  
90      @staticmethod
91      def output_menu_tasks():
92          """  Display a menu of choices to the user
93  
94          :return: nothing
95          """
96          print('''
97          Menu of Options
98          1) Add a new Task
99          2) Remove an existing Task
100         3) Save Data to File
101         4) Exit Program
102         ''')
103         print()  # Add an extra line for looks
104 
105     @staticmethod
106     def input_menu_choice():
107         """ Gets the menu choice from a user
108 
109         :return: string
110         """
111         choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
112         print()  # Add an extra line for looks
113         return choice
```

Figure 12. The IO class for performing input and output takes is in line 87. The first function in this class, output_menu_tasks is seen in line 91-103. The second function, input_menu_choice is found in lines 106-113.

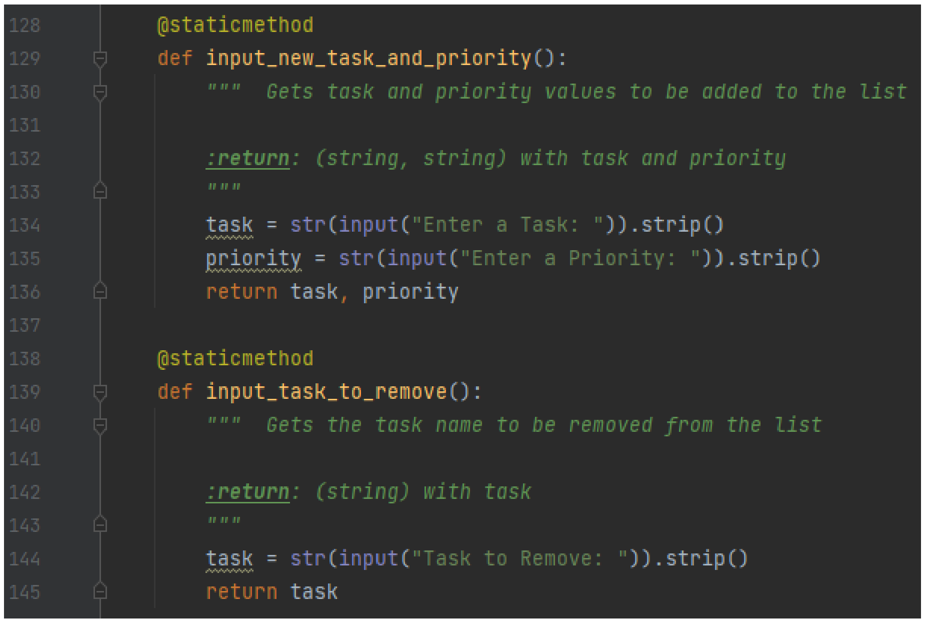
Next, we need a function to display the current tasks in the list. This function will be called output_current_tasks_in_list. How to write this code block is shown in Figure 13, below.



```
    @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 on your To Do List are:")
        for row in list_of_rows:
            print(row["Task"] + " (" + row["Priority"] + ")")
        print("********************************************")
        print()  # Add an extra line for looks
```

Figure 13. How to create a function to show current tasks in the list of dictionary rows.

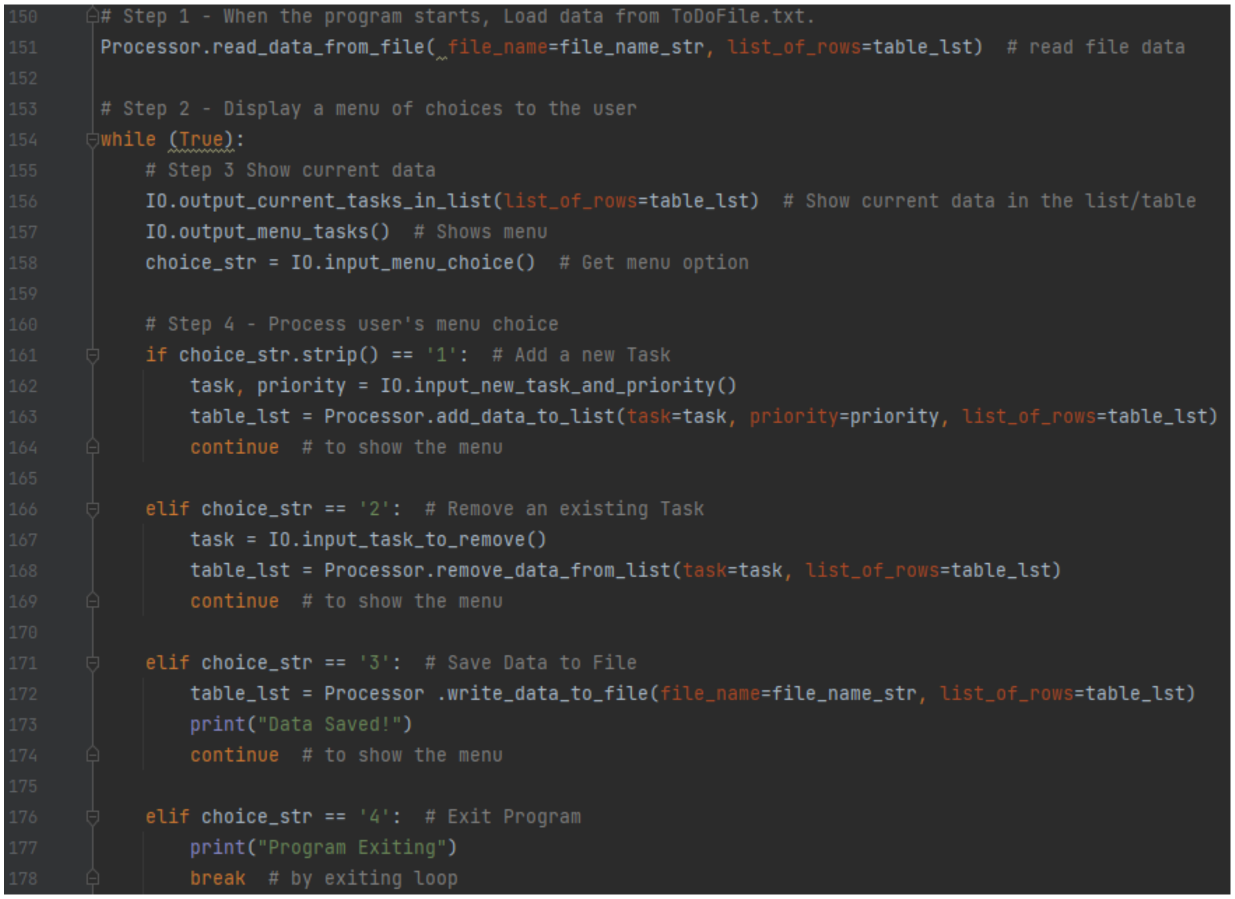
Next, we need two more functions, one to gather the input needed to add new tasks and priorities and one to gather the user input on what tasks to remove. Both functions can be found in Figure 14.



```
128     @staticmethod
129     def input_new_task_and_priority():
130         """  Gets task and priority values to be added to the list
131 
132         :return: (string, string) with task and priority
133         """
134         task = str(input("Enter a Task: ")).strip()
135         priority = str(input("Enter a Priority: ")).strip()
136         return task, priority
137 
138     @staticmethod
139     def input_task_to_remove():
140         """  Gets the task name to be removed from the list
141 
142         :return: (string) with task
143         """
144         task = str(input("Task to Remove: ")).strip()
145         return task
```

Figure 14. Lines 129-136 show the function, input_new_task_and_priority, which collects user input on what task to add and what the priority is. Lines 139-145 show the function, input_task_to_remove, which gets the task name that the user wants to be removed.

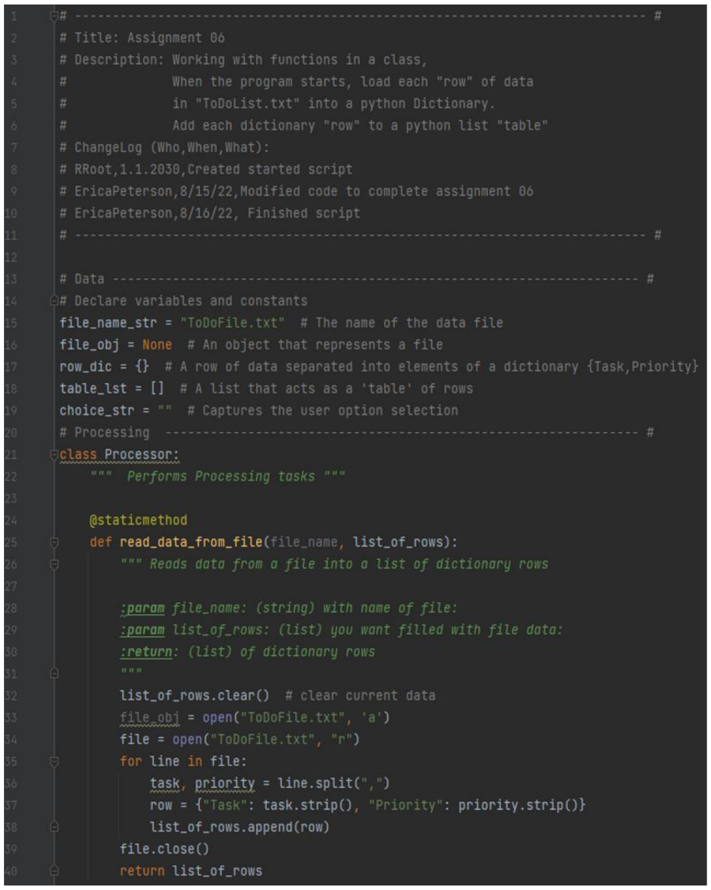
Finally, we need to add the two classes, processor and IO, together to run a script that can create a ToDoFile.txt file, add new tasks, remove tasks, save the data to a file, and exit. This is accomplished by using while loop with an if-elif statement inside of it to cycle through each menu choice. The code block to do this is seen in Figure 15, on the next page.



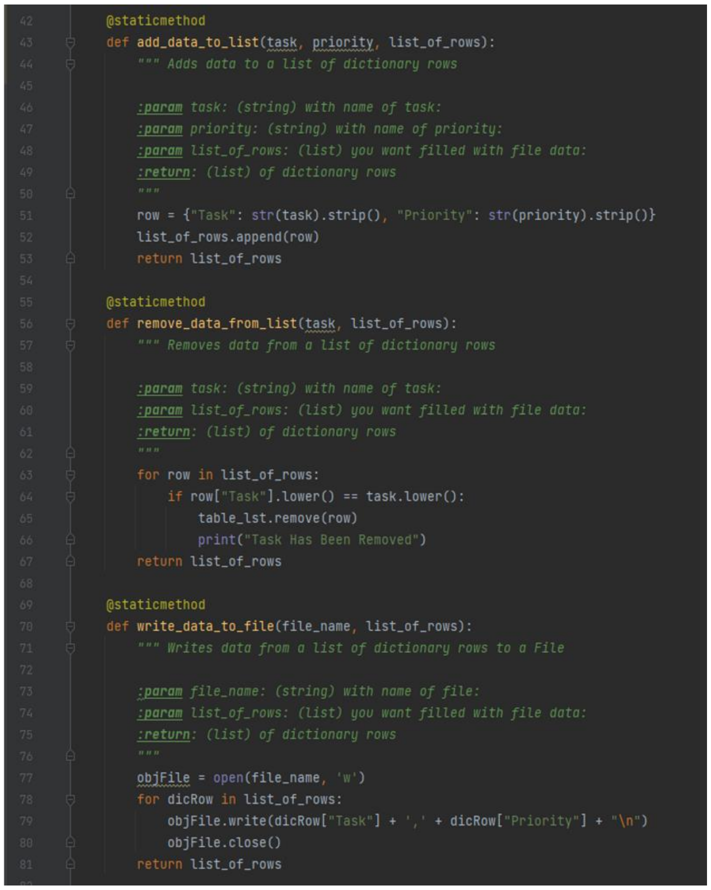
```python
# Step 1 - When the program starts, Load data from ToDoFile.txt.
Processor.read_data_from_file( file_name=file_name_str, list_of_rows=table_lst)  # read file data

# 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, priority = IO.input_new_task_and_priority()
        table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '2':  # Remove an existing Task
        task = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(task=task, 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("Program Exiting")
        break  # by exiting loop
```

Figure 15. The main body of the script which adds the two functions together to accomplish a ToDoList file. This code block was provided by Professor Randal Root and I made minimal changes to it.

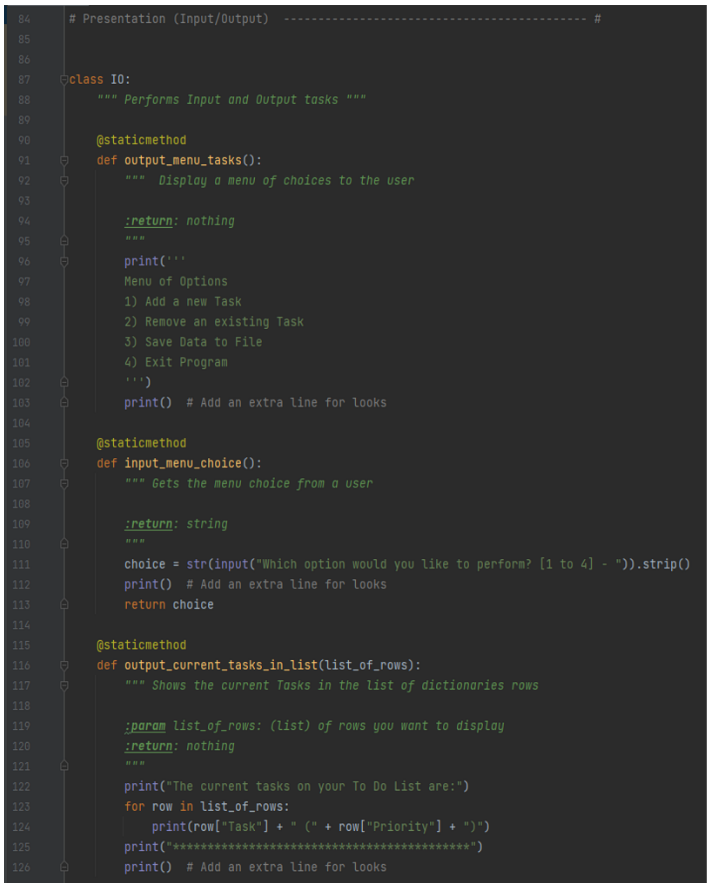
Finally, I will show all the code together so that you can see how it is all organized and how it meshes together. The complete script in PyCharm can be seen on the following page in Figure 16. This script running in PyCharm will be on page 11, in Figure 17.



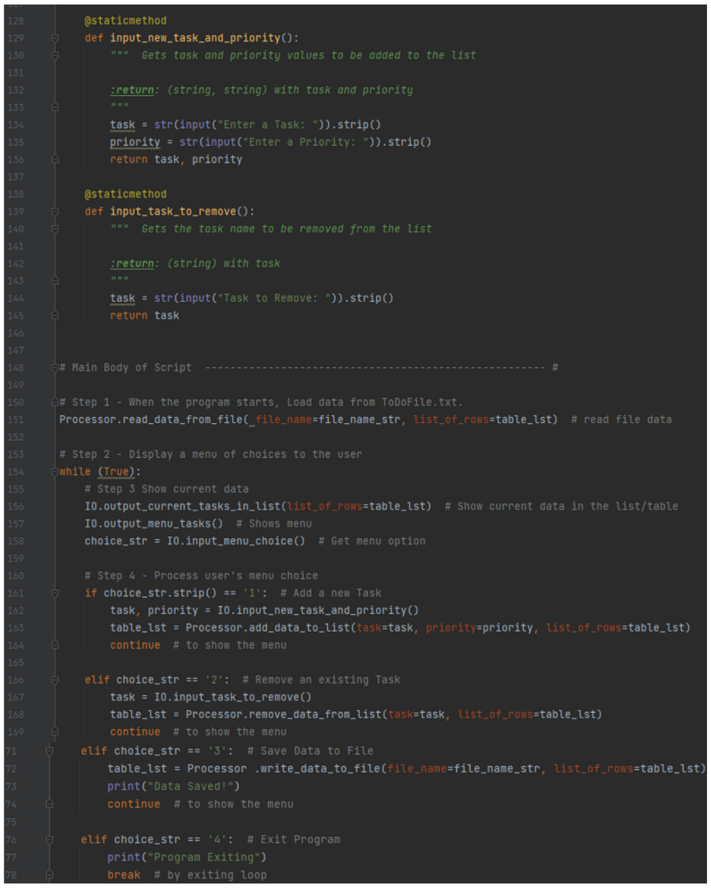
```python
# ---------------------------------------------------------------------------- #
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoList.txt" into a python Dictionary.
#              Add each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# EricaPeterson,8/15/22,Modified code to complete assignment 06
# EricaPeterson,8/16/22, Finished script
# ---------------------------------------------------------------------------- #

# Data ---------------------------------------------------------------------- #
# Declare variables and constants
file_name_str = "ToDoFile.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,Priority}
table_lst = []  # A list that acts as a 'table' of rows
choice_str = ""  # Captures the user option selection
# Processing  --------------------------------------------------------------- #
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
        file_obj = open("ToDoFile.txt", 'a')
        file = open("ToDoFile.txt", "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
```



```python
    @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

    @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():
                table_lst.remove(row)
                print("Task Has Been Removed")
        return list_of_rows

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

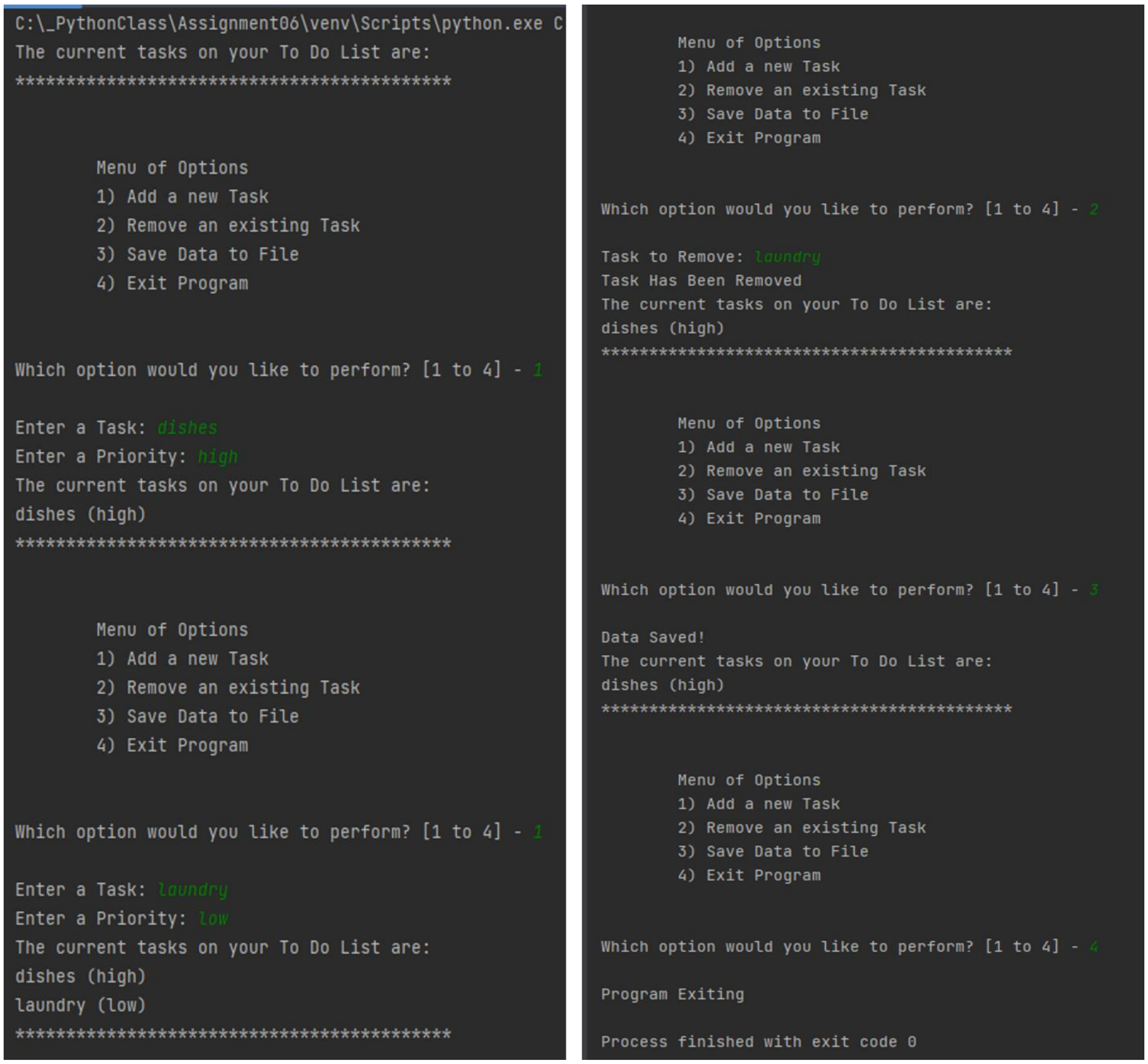


```python
# Presentation (Input/Output)  -------------------------------------------- #


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

    @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

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

    @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 on your To Do List are:")
        for row in list_of_rows:
            print(row["Task"] + " (" + row["Priority"] + ")")
        print("*******************************************")
        print()  # Add an extra line for looks
```



```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 a Priority: ")).strip()
        return task, priority

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

        :return: (string) with task
        """
        task = str(input("Task to Remove: ")).strip()
        return task


# 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, priority = IO.input_new_task_and_priority()
        table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '2':  # Remove an existing Task
        task = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(task=task, 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("Program Exiting")
        break  # by exiting loop
```

Figure 16. The completed script for creating a To Do List.



```
C:\_PythonClass\Assignment06\venv\Scripts\python.exe C
The current tasks on your To Do List are:
*******************************************


        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: dishes
Enter a Priority: high
The current tasks on your To Do List are:
dishes (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: laundry
Enter a Priority: low
The current tasks on your To Do List are:
dishes (high)
laundry (low)
*******************************************
```

```
        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

Task to Remove: laundry
Task Has Been Removed
The current tasks on your To Do List are:
dishes (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] - 3

Data Saved!
The current tasks on your To Do List are:
dishes (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

Program Exiting

Process finished with exit code 0
```

Figure 17. The completed script running in PyCharm, including user inputs in green.

Summary

In this assignment, I went over functions, classes, and how to use them to organize code. I also went over an example using classes and functions to create a script that manages a to do list. This will be extremely useful as we move on to designing more complex programs in future assignments.