

Janelle Bernales

May 24, 2023

Intro to Programming (Python)

Assignment 06

<https://github.com/JBernale/IntroToProg-Python.git>

Intro to Programming: Using Functions and Classes

Introduction

In Module 06 I learned how to use functions and classes to make a script easier-to-understand and edit. I continued to build on the script from Module 06 that we used to create a simple text-based program that records, displays, and edits a user's input prioritized "to do" tasks and saves this list to a text file. While this version of the program functions similarly to the previous version, the updated code uses classes which allow the program to call specifically grouped functions, parameters, and variables to perform a task.

Using classes can help make your code more organized, reusable, and user-friendly-particularly if someone else needs to edit it.¹

Creating the Program

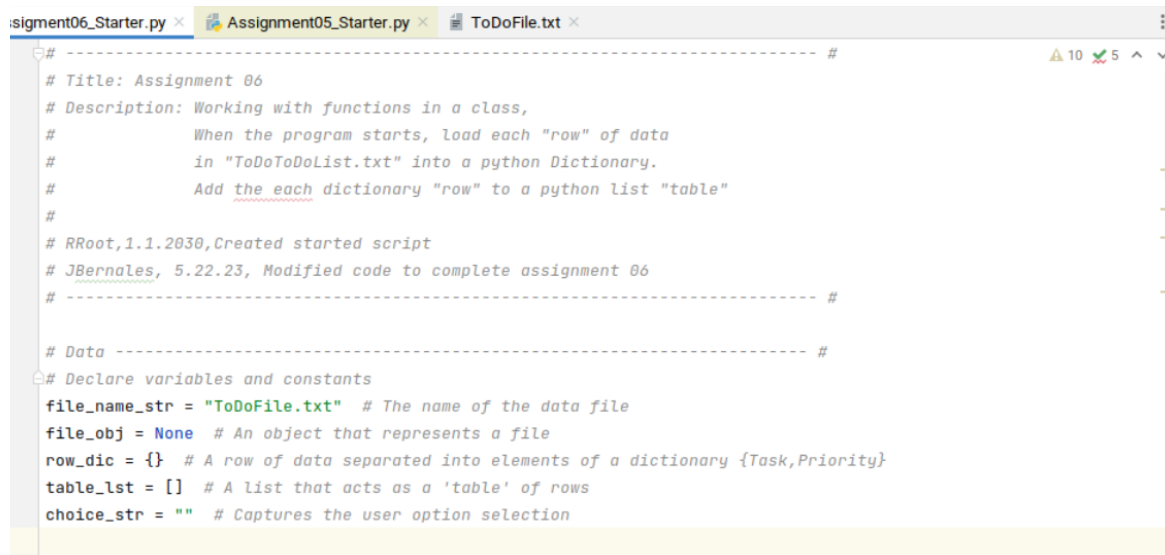
First, I uploaded the provided script for Module 06 titled "Assignment06_Starter.py" within the "Assignment06" subfolder in my "_PythonClass" folder. I also created a new project in PyCharm titled "Assignment06."

¹ "Python Classes: The Power of Object-Oriented Programming." Pozo Ramos, Leodanis, 2023.

Accessed May 24, 2023.

<https://realpython.com/python-classes/#understanding-the-benefits-of-using-classes-in-python>

Once I added the script to my PyCharm project, I updated the notes in the script header. The global variables, or variables that are referenced throughout the script, are already defined under the script header:



```
# ----- #
# Title: Assignment 06
# Description: Working with functions in a class,
#             When the program starts, load each "row" of data
#             in "ToDoToDoList.txt" into a python Dictionary.
#             Add the each dictionary "row" to a python list "table"
#
# RRoot,1.1.2030,Created started script
# JBernaies, 5.22.23, Modified code to complete assignment 06
# ----- #

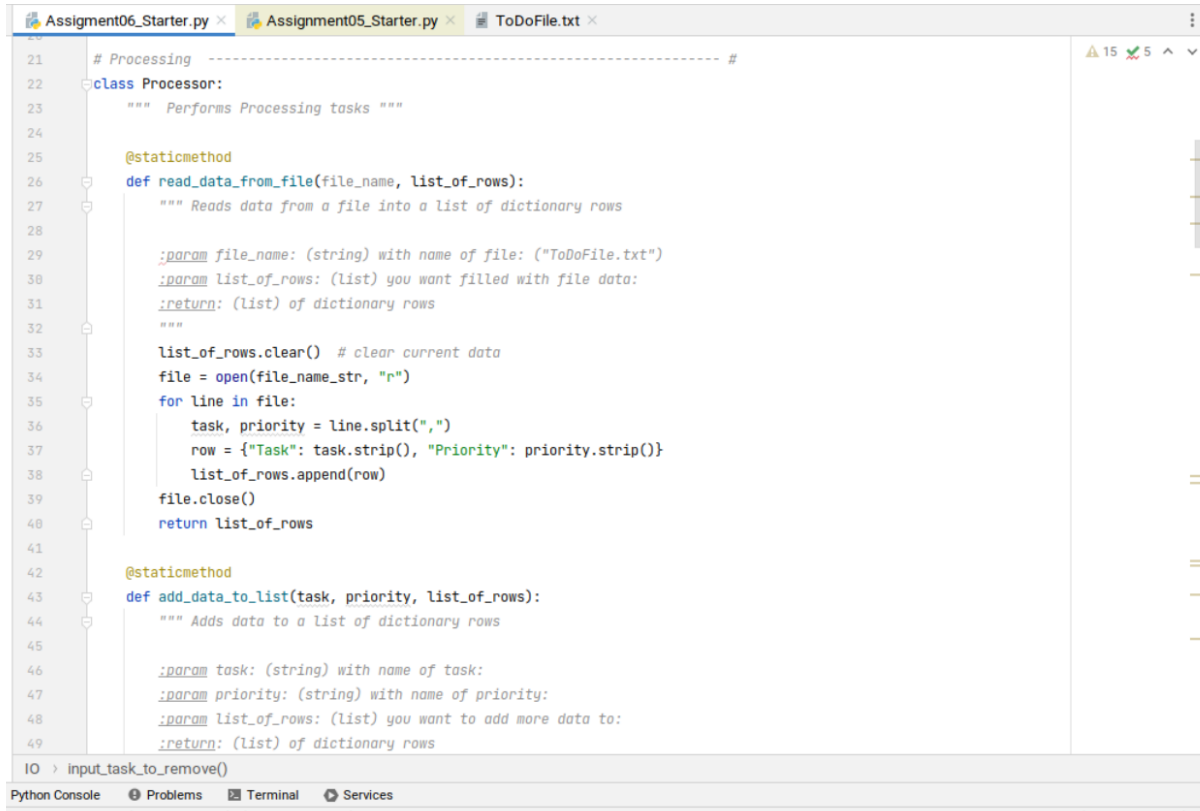
# Data ----- #
# Declare variables and constants
file_name_str = "ToDoFile.txt" # The name of the data file
file_obj = None # An object that represents a file
row_dic = {} # A row of data separated into elements of a dictionary {Task,Priority}
table_lst = [] # A list that acts as a 'table' of rows
choice_str = "" # Captures the user option selection
```

Fig. 1: The script header notes followed by global variables that have already been defined.

Finish defining functions in the “Processor” section

Next, I added some code to complete the class objects defined in the “Processing” class section of the code. In the starter code for Module 06, the script is written in three clearly defined sections: Processing, IO (Input/Output), and Main Body. Organizing your code within sections related to its purpose helps make it easier for others to understand what each grouped block of code does within the program.

The functions within the Processor class each contain code used to either read, write, remove, or save the user’s input to the “ToDoList.txt” file:



```
21 # Processing ----- #
22 class Processor:
23     """ Performs Processing tasks """
24
25     @staticmethod
26     def read_data_from_file(file_name, list_of_rows):
27         """ Reads data from a file into a list of dictionary rows
28
29         :param file_name: (string) with name of file: ("ToDoFile.txt")
30         :param list_of_rows: (list) you want filled with file data:
31         :return: (list) of dictionary rows
32         """
33         list_of_rows.clear() # clear current data
34         file = open(file_name_str, "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
41
42     @staticmethod
43     def add_data_to_list(task, priority, list_of_rows):
44         """ Adds data to a list of dictionary rows
45
46         :param task: (string) with name of task:
47         :param priority: (string) with name of priority:
48         :param list_of_rows: (list) you want to add more data to:
49         :return: (list) of dictionary rows
50         """
51
52 IO > input_task_to_remove()
```

Fig. 2: Defining functions within the Processor class allows us to use these functions and their associated parameters elsewhere in the code, without having to repeat the block of code multiple times.

Build out the “Presentation” section

In the following “Presentation” section, I completed code related to the “I/O,” or “Input/Output,” class of the program. All blocks of code within this class are used to produce the text that the user sees while interacting with the program, such as adding or deleting tasks and their associated priorities, saving their input to the text file, and exiting the program:



```

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

')
        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() # Select action to perform
        print() # Add an extra line for looks
        return choice # Executes the selected menu option

```

Fig. 3: The functions within the “IO” class contain code related to user interactions that can be called multiple times throughout the script

Finishing the code within the “IO” class was the most challenging part of this module for me. I kept getting an error message related to referenced variables. Usually this was because of a simple typo, like using the incorrect case when typing a variable name or referencing the wrong variable. This error reminded me of the importance of carefully double-checking your code, line-by-line, and making sure you reference variables consistently:

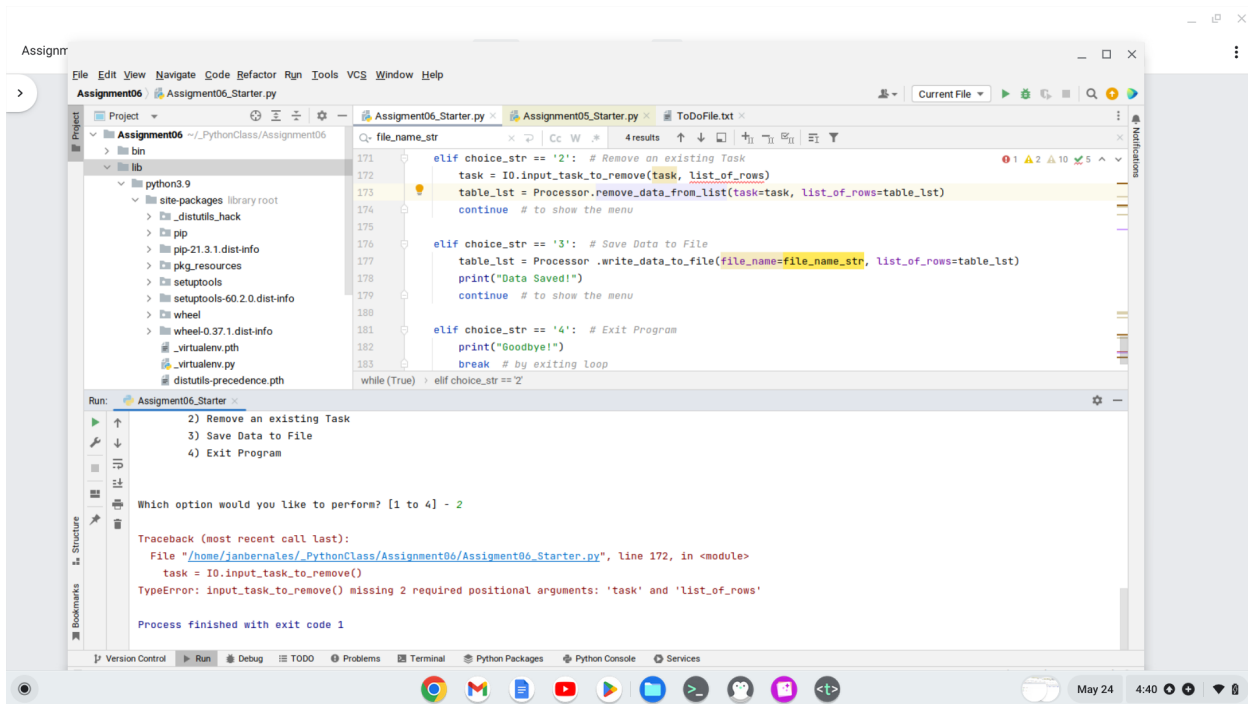


Fig. 4: TypeError caused by referencing variables incorrectly

After I sorted out what was causing the TypeError building out the rest of this section was fairly straightforward. I defined the variables “task” and “priority” with the appropriate user input as referenced in the “input_new_task_and_priority” and “input_task_to_remove” functions:

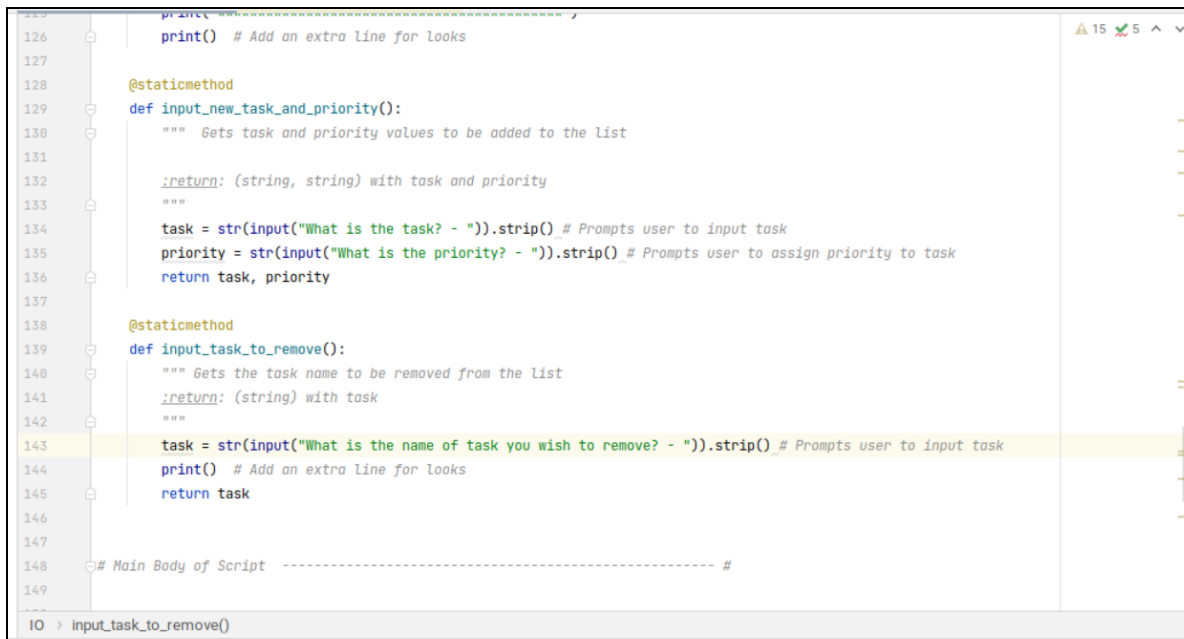
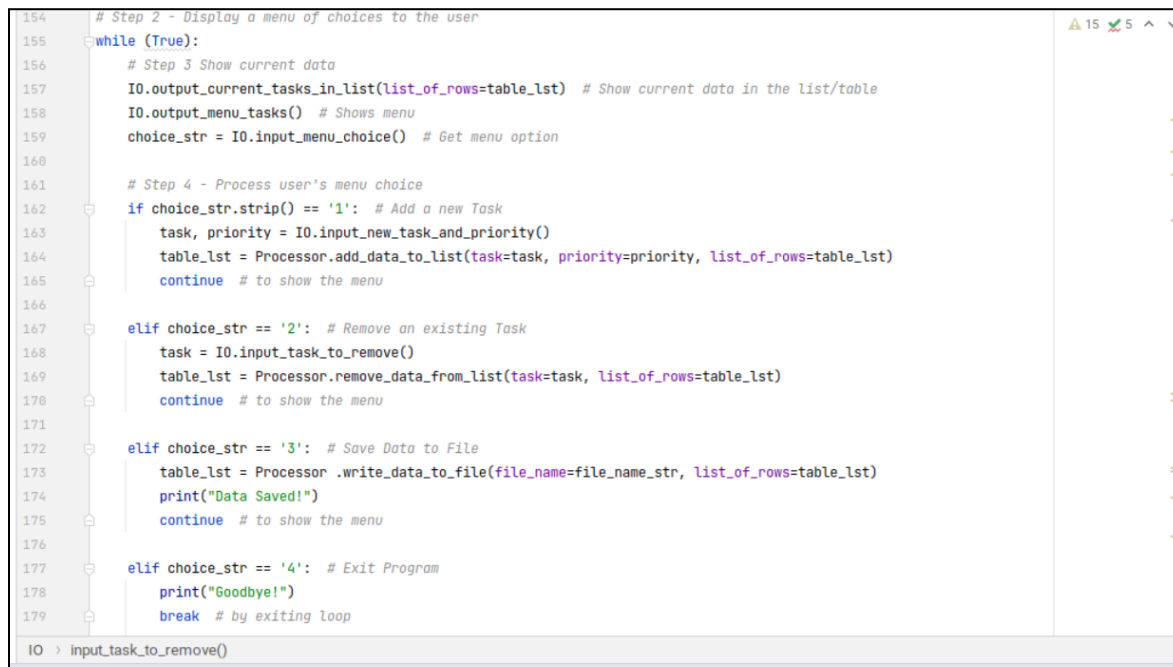


Fig. 5: Completing the code for the “input_new_task_and_priority” and “input_task_to_remove” functions with defined variables containing user input

Main Body of the Script

With the “Processor” and “I/O” sections complete, the script can now reference the functions and their related variables defined in their respective sections. Instead of having to repeat the same blocks of code over and over again to complete an action, such as adding a new task and priority, using classes allows us to call a group of specific functions using the function name.

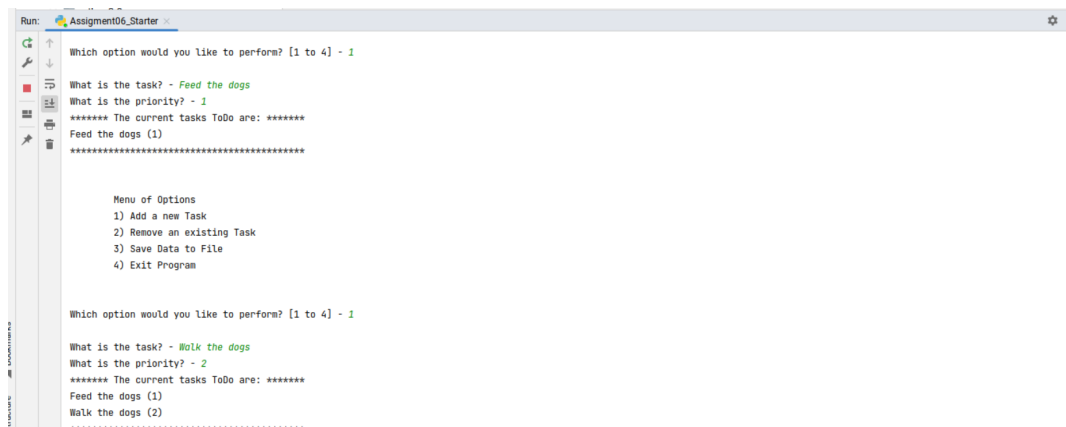
This makes the script more streamlined and easier to understand:



```
154 # Step 2 - Display a menu of choices to the user
155 while (True):
156     # Step 3 Show current data
157     IO.output_current_tasks_in_list(list_of_rows=table_lst) # Show current data in the list/table
158     IO.output_menu_tasks() # Shows menu
159     choice_str = IO.input_menu_choice() # Get menu option
160
161     # Step 4 - Process user's menu choice
162     if choice_str.strip() == '1': # Add a new Task
163         task, priority = IO.input_new_task_and_priority()
164         table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
165         continue # to show the menu
166
167     elif choice_str == '2': # Remove an existing Task
168         task = IO.input_task_to_remove()
169         table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
170         continue # to show the menu
171
172     elif choice_str == '3': # Save Data to File
173         table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
174         print("Data Saved!")
175         continue # to show the menu
176
177     elif choice_str == '4': # Exit Program
178         print("Goodbye!")
179         break # by exiting loop
180
181 IO > input_task_to_remove()
```

Fig. 6: The Main Body section of the script uses functions defined within the “Processor” and “I/O” classes to reference their associated variables.

After adding references to the appropriate functions, I ran the program in Pycharm and Terminal:



```
Run: Assignment06_Starter
Which option would you like to perform? [1 to 4] - 1
What is the task? - Feed the dogs
What is the priority? - 1
***** The current tasks ToDo are: *****
Feed the dogs (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
What is the task? - Walk the dogs
What is the priority? - 2
***** The current tasks ToDo are: *****
Feed the dogs (1)
Walk the dogs (2)
*****
```

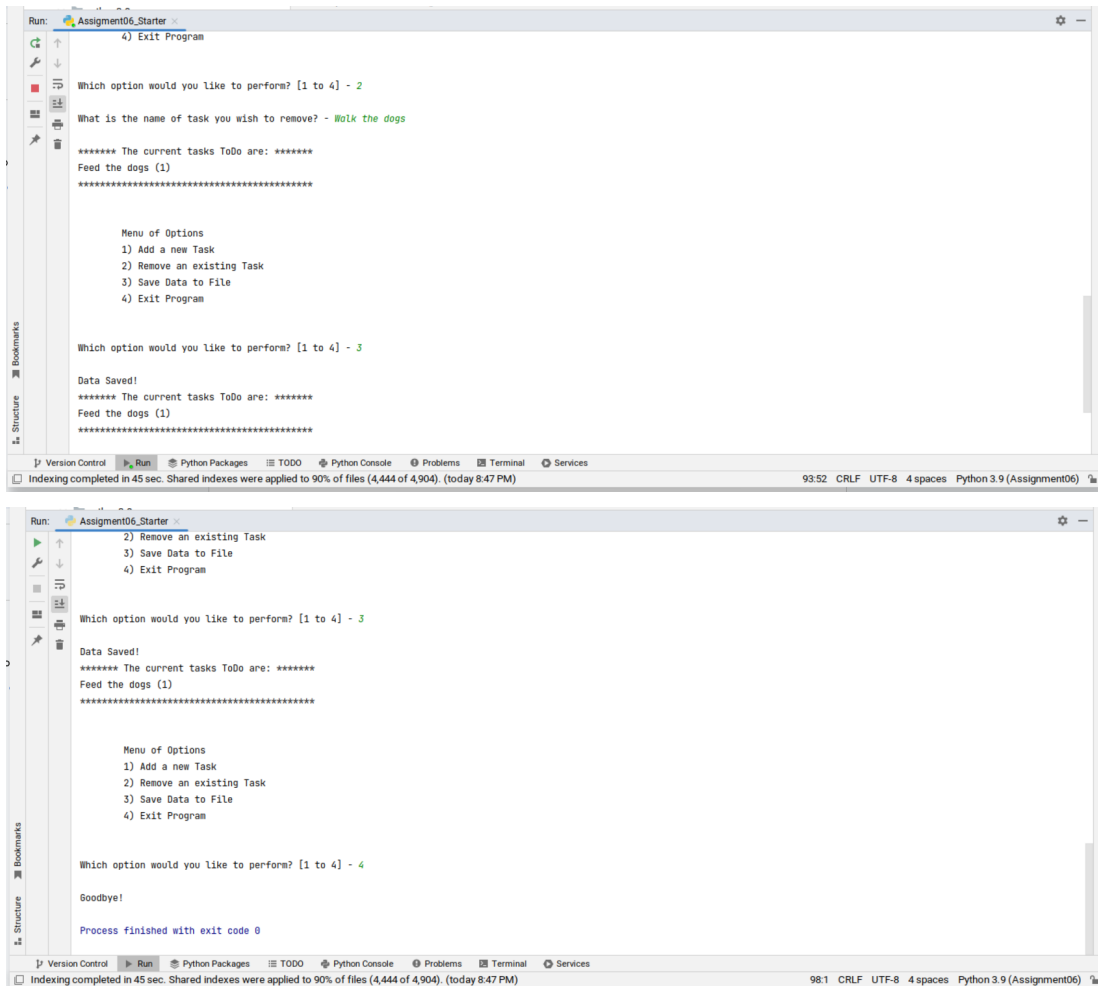


Fig. 7: The program has successfully run in Pycharm.

Summary

In Module 06 I learned how using classes to call grouped blocks of code and their associated functions and variables can make your code much easier to read. Although it was tricky understanding how local versus global variables worked, I had a better understanding of how these variables function differently throughout a script. In addition, I learned:

- **Building on someone else's code, while sometimes challenging, can be very helpful to learn how to cleanly organize your own code.** Having someone else's code to reference as a template or starting off point is extremely helpful, especially to a new programmer who is not yet accustomed to the standard formatting conventions of a language.
- **Classes make it easier to reuse and repurpose code.** Not only do classes make your script cleaner by allowing you to reference a group of associated functions and variables, they also make it easier to identify at a glance what a particular block of code does. This

also makes it easier to know whether a particular class can do the task you're looking to accomplish so you can reuse it the same way in another program.