Tim Shore
November 23, 2020
Foundations of Programming: Python
Assignment 06
https://github.com/timothyshore/IntroToPython

Using Classes and Functions in Task/Priority Program

Introduction

By implementing Classes and Functions into Python code, we can script our code in the processing section of the program and then call it to action in the presentation section. By using the Separation of Concerns (SoC) in the code, it facilitates a much easier guide to read and to make any necessary changes. With well defined classes and functions, we will be able to call upon them to process multiple variables by passing them into the values.

Functions

There are invisible functions that are already built into Python (such as **abs()** to get the absolute value of a number) and functions that a programmer can create themselves. To create a function, the code is **def** for define followed by a space and the name you give the function followed by parentheses. Now that the custom function has been created, it can be scripted further in the processing section and then called in the presentation section - just like you would call a built in function with the name followed by parentheses.

Parameters and Arguments

Like the functions already built into Python, you can pass code in the parentheses. With **abs**(), you pass the integer you want to know the absolute value of in between the parentheses – *example: abs(-7) is 7*. With a custom function, you first create the parameter of the function when defining it. Then you would script what value that parameter would hold. When you call the function to action, the parameter is passed onto the argument that is inside the parentheses. For example:

```
def DogTalk(bark)
    print(bark)

DogTalk(“woof woof”)
```

What would be printed is: woof woof. This is because the parameter was defined with a value and that value would be printed. When we called the function, the parameter was replaced with the argument. This is helpful when you want to create a code that is called to action several times, but you want to pass in different arguments when the function runs.

Return Values

You can also define the value you want to call up when scripting the function. At the end of the script in creating the function, you use the **return** statement and the value that you want the function to pass to the variable when the function is called to action. Using the function this way can allow you to write one function and use it multiple times with different inputs from the user if the process is the same across different actions.

Classes

Not only can you use functions to encapsulate or wrap some of the program processing, you can also create a class to encapsulate the functions, variables and constants separately from other functions. This can be helpful when identifying which process is being called upon in the presentation portion of the code. A class is created much like a function, except you use the statement **class** instead of **def** at the start of the coding. Then before the functions are created (and indented to place them inside the class), you use the phrase **@staticmethod.** Then when you call up the functions in the presentation portion of the script, you first identify the class by name and then the function name separated by a period. For example: **ClassName.FunctionName().**

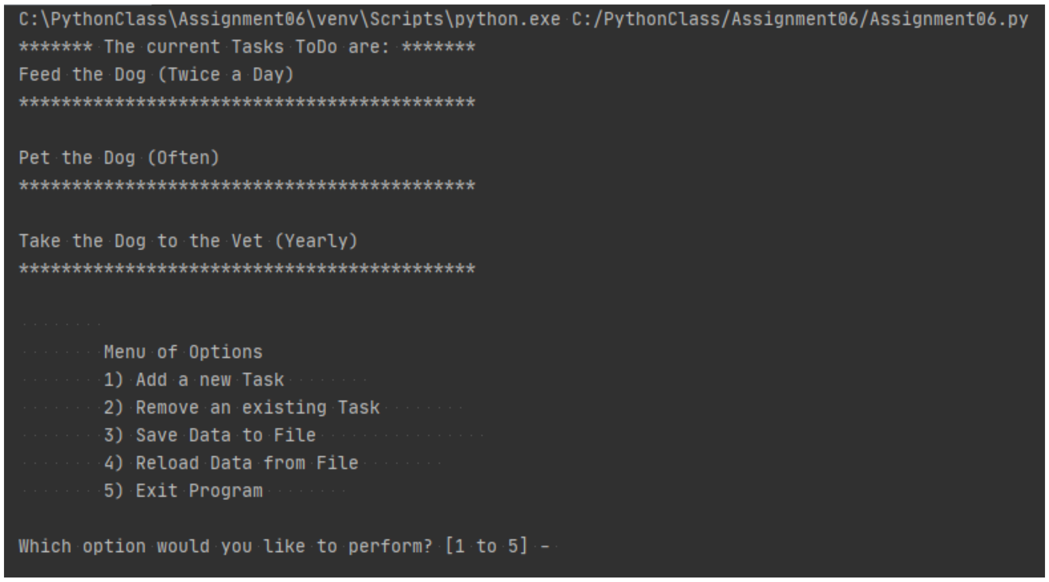
Scripting the Assignment

Using the Starter File

For this assignment, we were provided a starter file with indications of "To Do" portions where to insert code to make it work. At the suggestion of the instructor, I kept the a copy of the starter file separate from the file where I was writing my code. This made it much easier to refer to and see where I had added code and where the code was already there. The program is very similar to the last assignment, with the exception that most of the coding required was to be scripted in the functions.

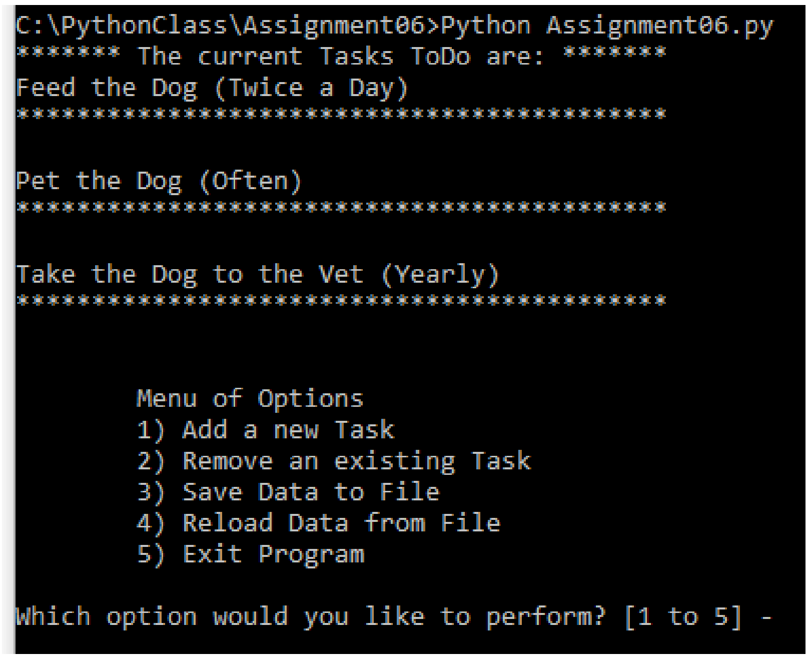
Data Loads When Starting the Program

This time around, I decided to create the text file outside of the program to simplify the necessary tasks. Here the code was already written. However, I had saved my text file under a different name then in the code and realized it as soon as it failed to read the text file. Here are the displays of the load in PyCharm (Figure 6.1) and the Command Line Prompt (Figure 6.2)



```
C:\PythonClass\Assignment06\venv\Scripts\python.exe C:/PythonClass/Assignment06/Assignment06.py
******* The current Tasks ToDo are: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************


        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 6.1: Loading the program in PyCharm



```
C:\PythonClass\Assignment06>Python Assignment06.py
******* The current Tasks ToDo are: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************



        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] -
```

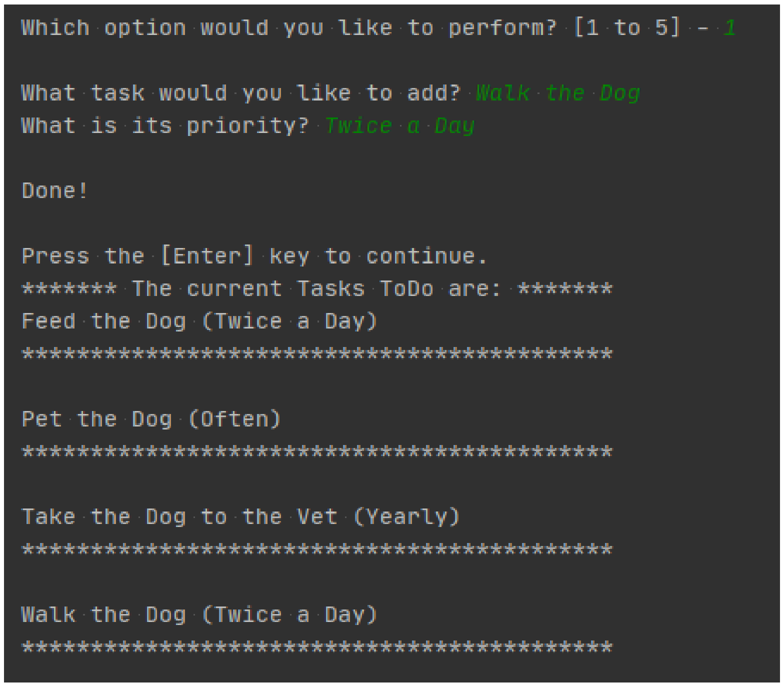
Figure 6.2: Loading the program in Command Line Prompt

Menu Option 1: Add a New Task

Now it was time to get down to brass tacks, and start coding a function. The name of the function and its three parameters were already coded; so was the return statement with the third parameter. So I could see that I needed to define that a task input and a priority input were passed into the list. Thanks to the videos that the instructor recorded, I could use basically the same programming I had used in the last assignment, but to trade out some of the variables I had scripted into the values defined in the parameters.

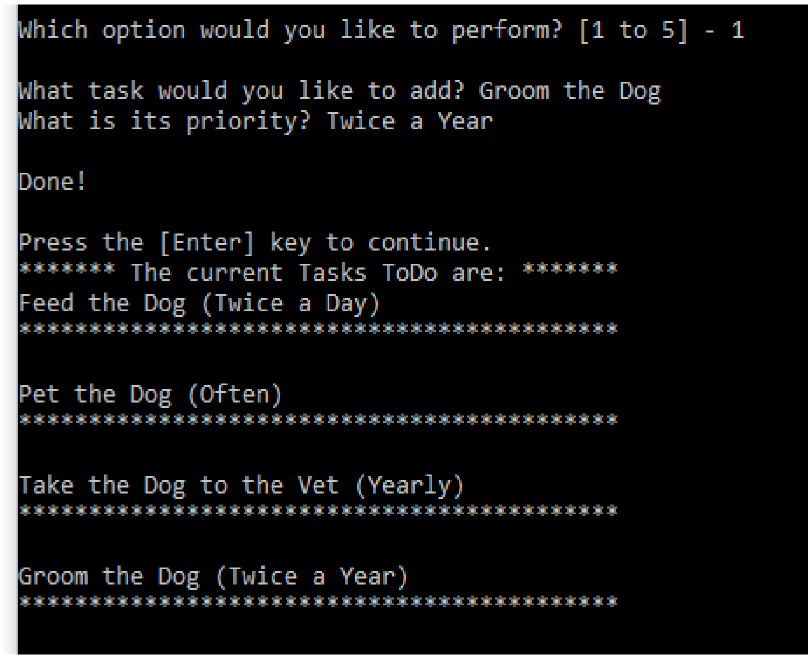
It was just a matter of separating the code into the correct processing functions. The first being the code to run the process in the Processor Class and the second being the code to run to ask the user to input their information in the IO Class.

However, it took me a while to realize that I needed to put my variables outside of the function code I was calling instead of putting inside the parenthesis as an argument. PyCharm was helpful in showing me that my values of task and priority were not being identified in the parameter. That's when I did some digging into the lessons and discovered where this was where the variables had to be passed into the function outside of the argument. This “a-ha” moment was probably the largest I had while struggling with the scripting of the code. Here are my successful programs in PyCharm (Figure 6.1) and the Command Line Prompt (Figure 6.2).



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

What task would you like to add? Walk the Dog
What is its priority? Twice a Day

Done!

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************

Walk the Dog (Twice a Day)
*******************************************
```

Figure 6.3: Adding to the List in PyCharm

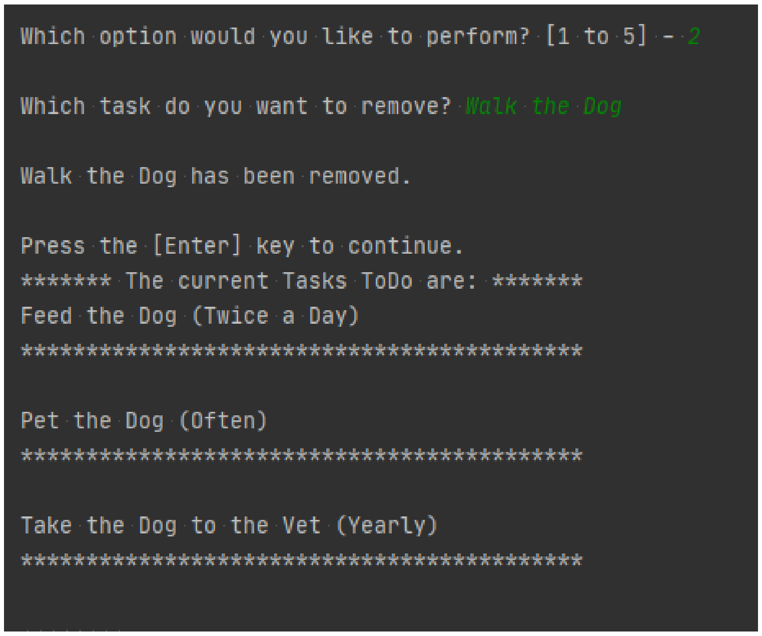


```
Which option would you like to perform? [1 to 5] - 1

What task would you like to add? Groom the Dog
What is its priority? Twice a Year

Done!

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************

Groom the Dog (Twice a Year)
*******************************************
```

Figure 6.4: Adding to the List in Command Line

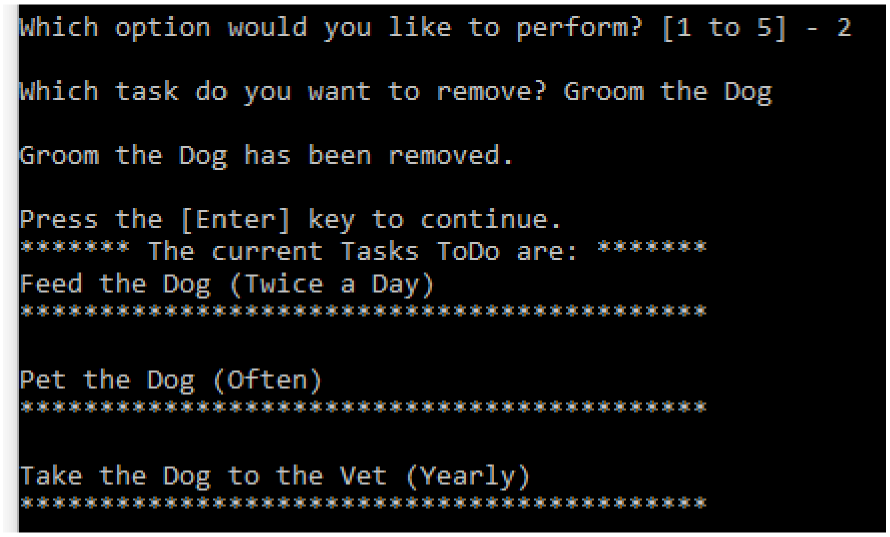
Menu Option 2: Removing an Existing Task

After struggling with making the first option work, this was a breeze. Once more, I used the code from the previous assignment and reassigned some variables for values in the script. One of the benefits I discovered when using a function for this task, was the ability to remove both the task and the priority by just calling up the task. My successful PyCharm (Figure 6.3) and Command Line (Figure 6.4) are below;



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

Which task do you want to remove? Walk the Dog

Walk the Dog has been removed.

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************
```

Figure 6.5: Removing a task from the data in PyCharm

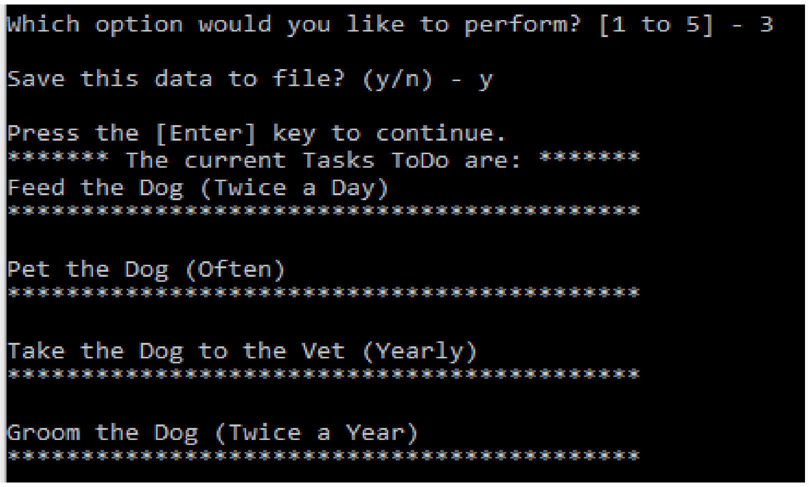


```
Which option would you like to perform? [1 to 5] - 2

Which task do you want to remove? Groom the Dog

Groom the Dog has been removed.

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************
```

Figure 6.6: Removing a task from the data in Command Line

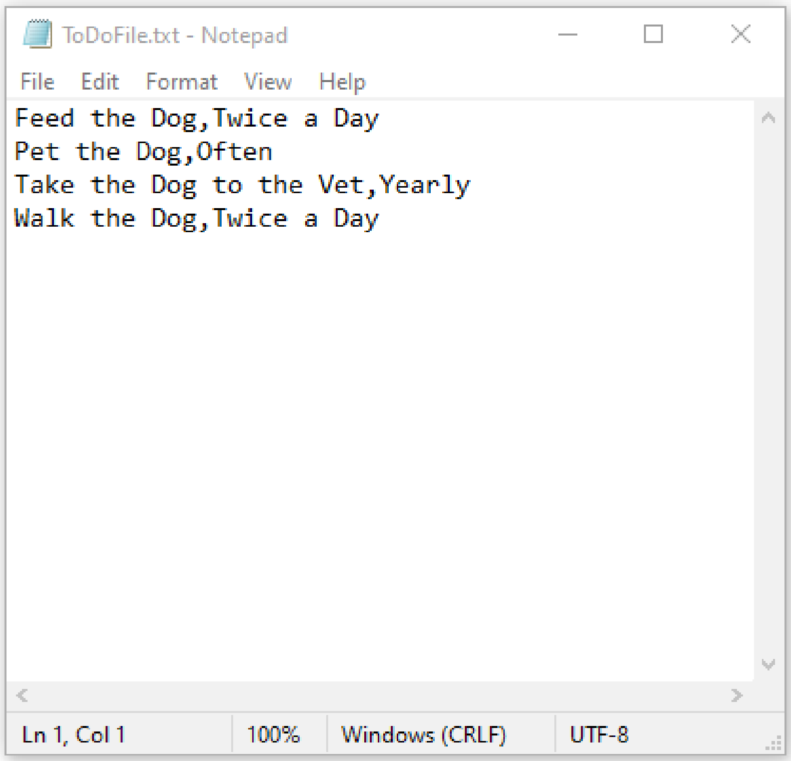
Menu Option 3: Save Data to File

Before I removed my task, I saved my data to the text file. That way I could see the difference between my initial file and the saved version. This was a breeze after struggling through the first menu option. My successful program runs in PyCharm and Command Line Prompt (Figure 6.7) and the result in the text file in PyCharm (Figure 6.8) and Command Line Prompt (Figure 6.9).



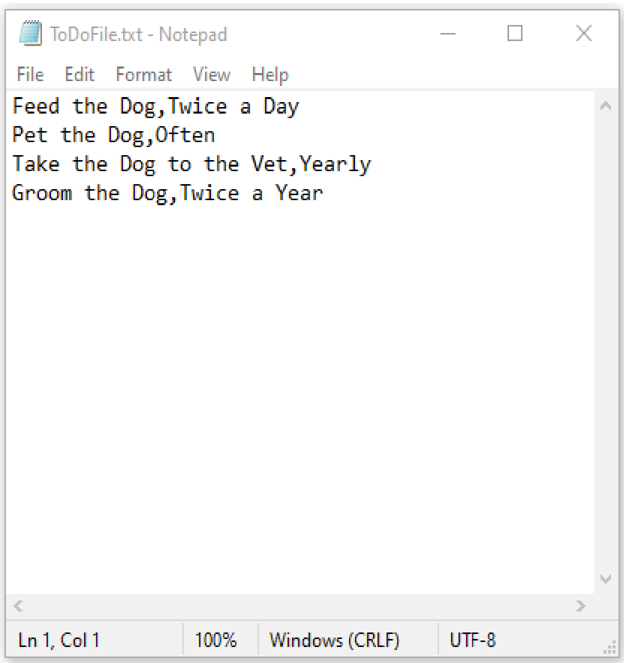
```
Which option would you like to perform? [1 to 5] - 3

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

Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************

Groom the Dog (Twice a Year)
*******************************************
```

Figure 6.7: Code could not find task in PyCharm



```
Feed the Dog,Twice a Day
Pet the Dog,Often
Take the Dog to the Vet,Yearly
Walk the Dog,Twice a Day
```

Figure 6.8: Saved Text File from PyCharm



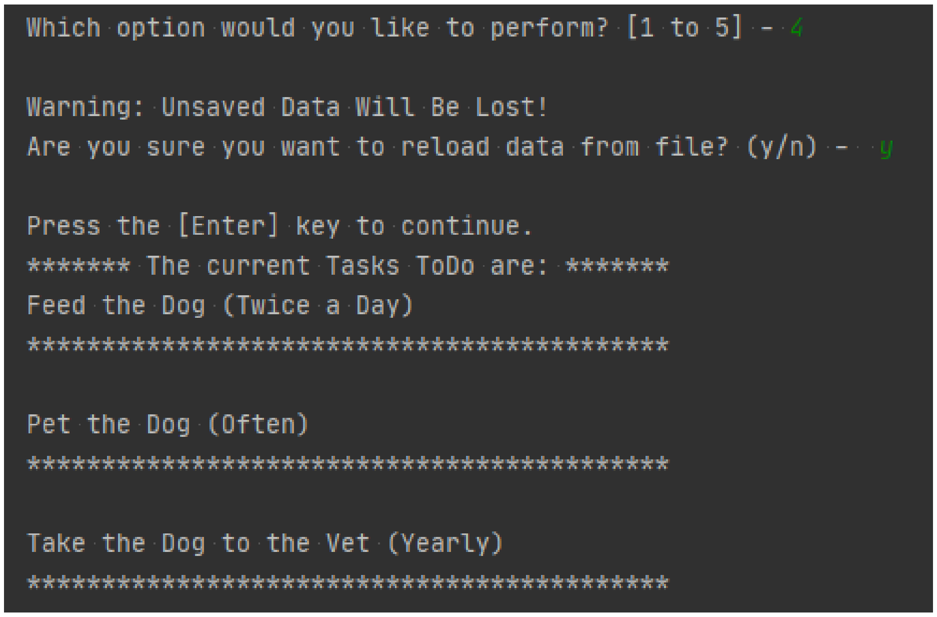
ToDoFile.txt - Notepad

```
Feed the Dog,Twice a Day
Pet the Dog,Often
Take the Dog to the Vet,Yearly
Groom the Dog,Twice a Year
```

Figure 6.9: Saved Text File from Command Line Prompt

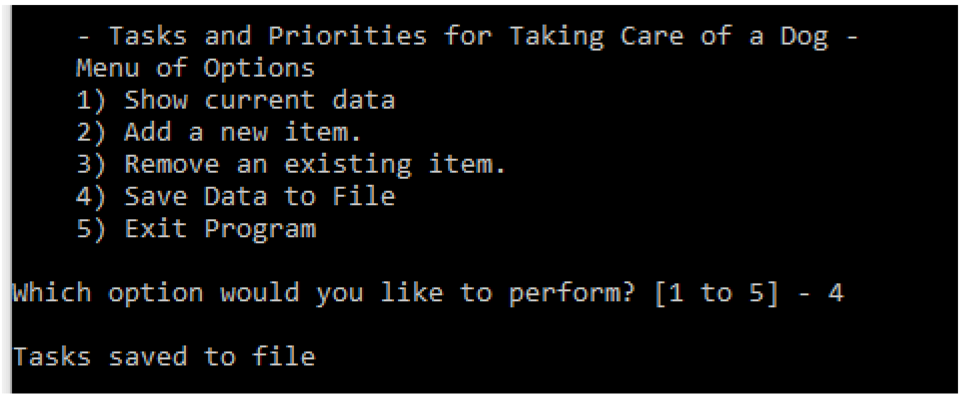
Menu Option 4: Reloading the Data from the Text File

For this menu option, it was a simple adding of the function code that runs from the opening of the program and calling it into user input 4 in the presentation section of the code. This was probably the best example of reusing a function to get a similar result without having to rewrite the code all over again. I did this menu option after removing the task and priority that I had added. Here are the successful runs of the program in PyCharm (Figure 6.10) and Command Line Prompt (Figure 6.11)



```
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: *******
Feed the Dog (Twice a Day)
*******************************************

Pet the Dog (Often)
*******************************************

Take the Dog to the Vet (Yearly)
*******************************************
```

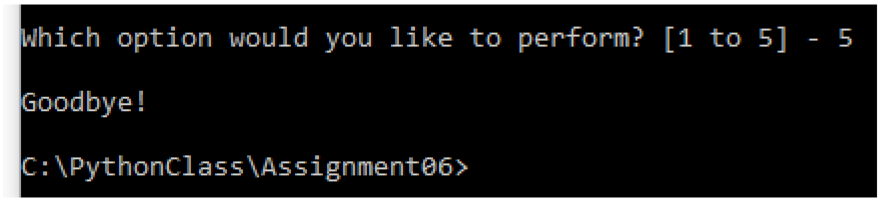
Figure 6.10: Data reloaded from text file to program in PyCharm



```
    - Tasks and Priorities for Taking Care of a Dog -
    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

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

Tasks saved to file
```

Figure 6.11: Data reloaded from text file to program in Command Line Prompt

Menu Option 5: Closing the Program



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

Goodbye!

C:\PythonClass\Assignment06>
```

Figure 6:12: The program closing after Option 5 in Command Line Prompt

Summary

The importance of creating functions and classes becomes self evident when creating a large program. Especially when creating a program where you would be calling upon a section of code to run again and again at different points in the presentation. It also allows for keeping the Separation of Concerns (SoC) in unique classes, so that the code is well organized and easy to read.