Susan Paulen

November 23, 2020

IT FDN 110 A

Assignment 06

GitHub: <https://github.com/spaulen/IntroToProg-Python-Mod06>

# The Home Inventory Program (Part 4)

# Introduction

This week, we learned about creating and manipulating functions, parameters, arguments, return values, global and local variables, static methods, as well as debugging tools in Python. Functions are helpful to organize code. Parameters are declared in a function while arguments are passed to a function when it is called. Global variables may be accessed anywhere in a program including functions. Local variables are declared within a function and for that reason can only be accessed within the function they are declared in. Return values are values returned from a function. The return statement immediately halts the function even if it isn’t the final statement in the function. Static methods are a method bound to a class, but they do not change the class. All of these Python objects, methods, parameters, arguments and statements are used in this week’s Assignment 06.

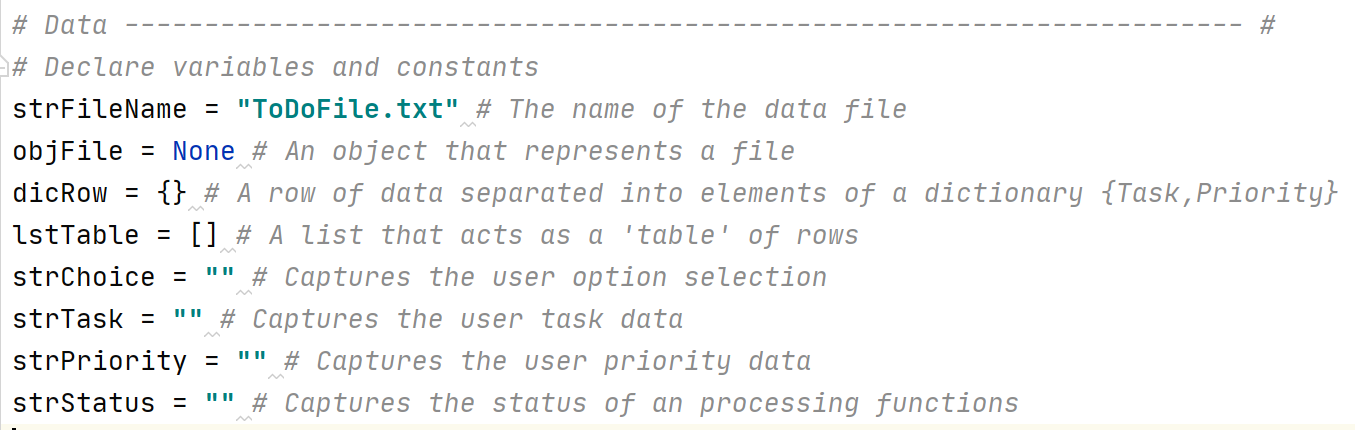
# Global Variable Declarations

As a best practice; file objects, strings, lists, list tables and dictionary rows are declared at the beginning of the Python code for this assignment. Both the List Table (lstTable) and the Dictionary Row (dicRow) should be declared before usage to avoid errors.

Explanations for each variable and constant are provided in Python comments identified by the hashtag below in Figure 1.1.

Note that lstTable is used to contain all changes made to the ToDoFile.txt data and will be saved over the ToDoFile.txt file at the end of a user session if the user chooses to do so. This week’s global variables were provided as part of the Assignment06\_Starter.py code.

Figure 1.1 Global Variables



## Classes & Program Script (Main Body)

Using Separation of Concerns appropriately, this program is separated into 3 parts:

1. Processor Class

This is where the file object rows are processed into a list object to avoid overwriting the file object while making use of its data. The four static methods in this section for read data, add data, remove data and write data use the parameters passed via call arguments in the program script. This is a good way to

1. IO Class

This is where input and output tasks are performed – capturing data from the user and passing back that data or printing information to the user display. The seven static methods in this section do not all return information via a return statement.

1. Program Script (Main Body)

This code contains the logic of the application (program). The program calls the processor class static methods as well as the IO class static methods to allow a user to read, add, remove and save their Home Inventory data.

## Processor Class & Static Methods

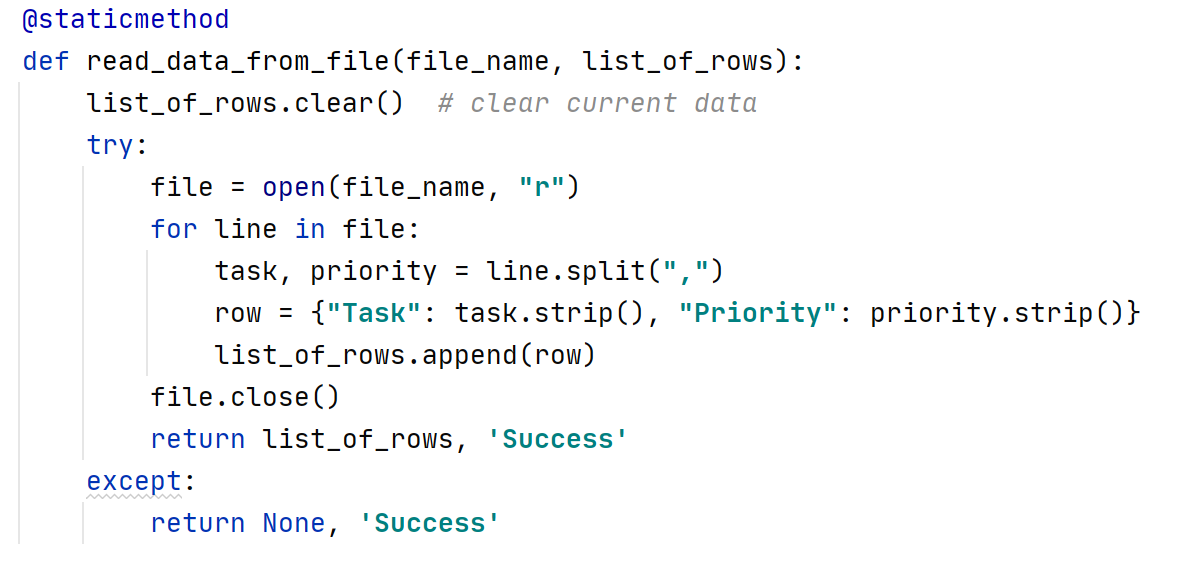
The class “Processor” was provided by the starter code for this assignment.

Static Methods are useful when the class only needs to be initialized once as we do in this program. The Processor class is broken into four static methods:

1. read\_data\_from\_file(file\_name, list\_of\_rows)

This method opens and reads a file object’s rows into a dictionary row. The dictionary rows are saved in a list. The parameters, (file\_name and list\_of\_rows), will be passed via arguments when this method is called. Based on the Try…Exists statements, whether the file exists or not, this method will return a status of Success.

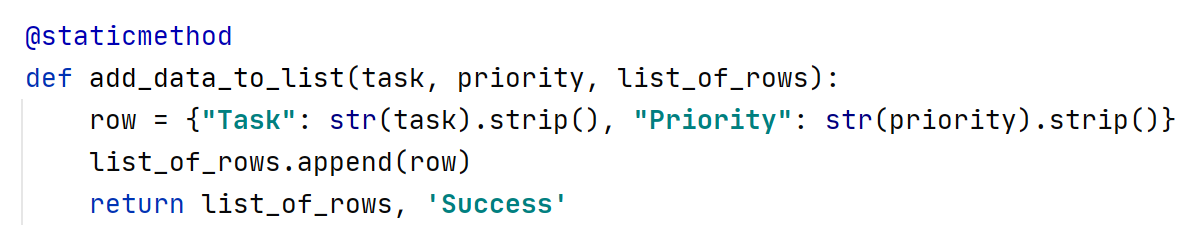
Figure .1 Static Method – read\_data\_from\_file



1. add\_data\_to\_list(task, priority, list\_of\_rows)

This method inserts one new item and its value into a dictionary row. The dictionary row is then appended to the list. The parameters (task, priority and list of rows), will be passed via arguments when this method is called. The full list is returned to the method call with a status of Success.

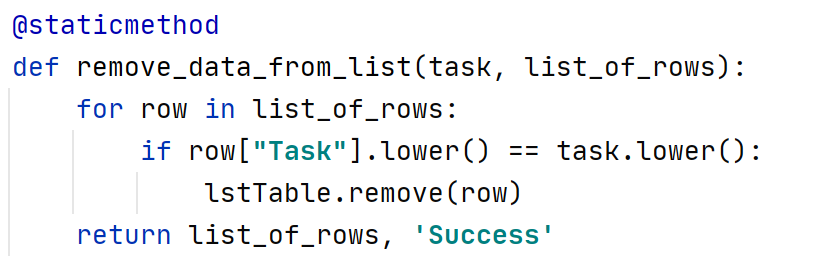
Figure .1 Static Method – add\_data\_to\_list



1. remove\_data\_from\_list(task, list\_of\_rows)

This method removes an item based on the Item Name (task). The list rows are search for the Item Name to remove using lower case in both the list row item name (task) and the task argument passed to the method task parameter. When the row is found, it is removed. The parameters (task and list of rows), will be passed via arguments when this method is called. The full list is returned to the program script call with a status of Success.

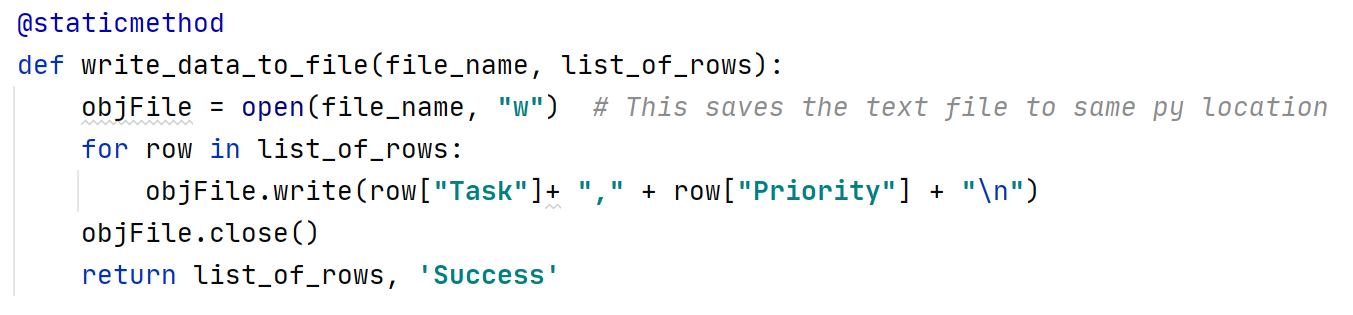
Figure 4.1 Static Method – remove\_data\_from\_list



1. write\_data\_to\_file(file\_name, list\_of\_rows)

This method writes the list\_of\_rows data to the file\_name in the same directory location as the py program file. The parameters (file\_name and list\_of\_rows), will be passed via arguments when this method is called. Each row in the list\_of\_rows is written to the file object and then the file object is closed. The full list is returned to the program script call with a status of Success.

Figure .1 Static Method – write\_data\_to\_file



## IO Class & Static Methods

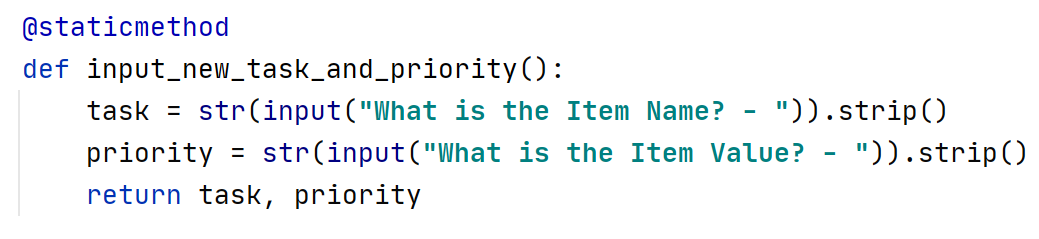
The class “IO” was provided by the starter code for this assignment.

Again, static methods are useful when the class only needs to be initialized once as we do in this program. The IO class is broken into seven static methods:

1. input\_new\_task\_and\_priority()

This method requests input from the user – task (Item Name) and priority (Item Value) and returns the input (task, priority) back to the calling statement in the program script.

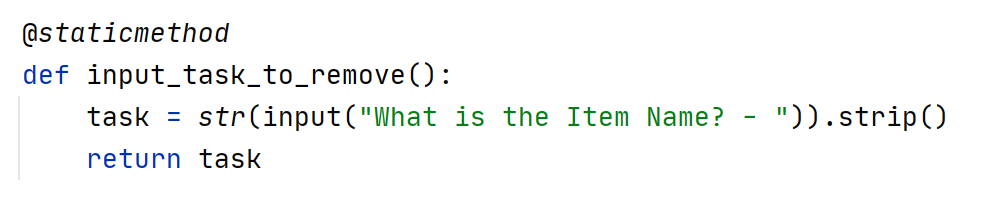
Figure .1 Static Method - input\_new\_task\_and\_priority



1. input\_task\_to\_remove()

This method requests input from the user – task (Item Name) and returns the input (task) back to the calling statement in the program script.

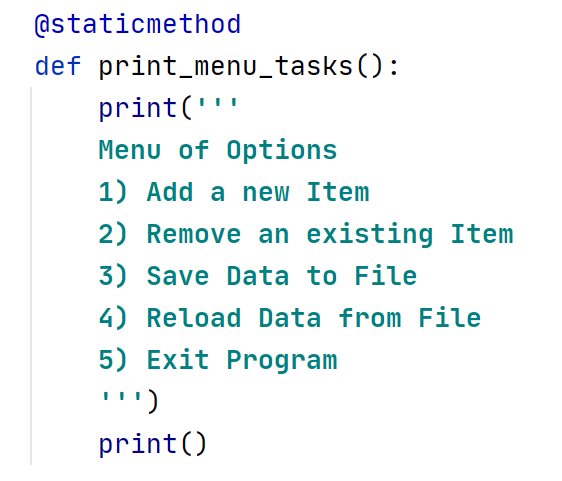
Figure .1 Static Method - input\_task\_to\_remove



1. print\_menu\_tasks()

This method displays the menu to the user.

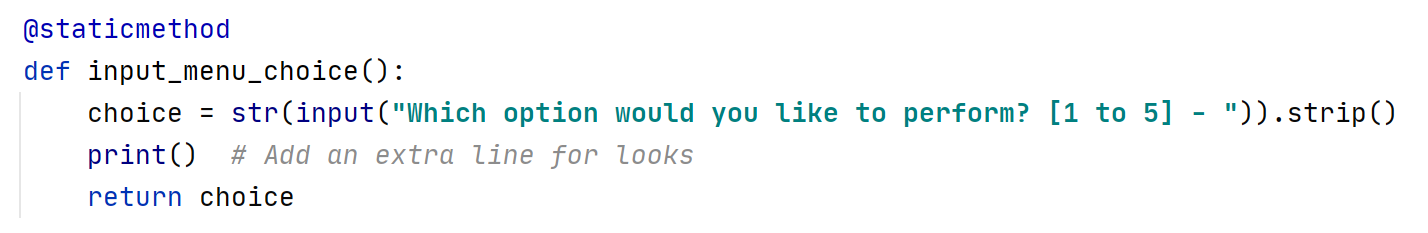
Figure .1 Static Method - print\_menu\_tasks



1. input\_menu\_choice():

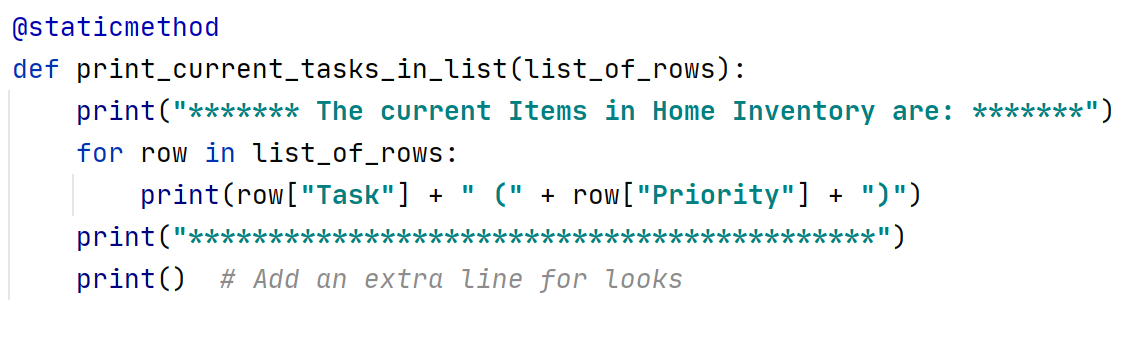
This method requests input from the user – choice (Menu Option choice selected) and returns the input (choice) back to the calling statement in the program script.

Figure .1 input\_menu\_choice



1. print\_current\_tasks\_in\_list()  
   This method displays the Item Names and Item Values in the current list object. This may differ from the existing file as the user may have added or removed rows.

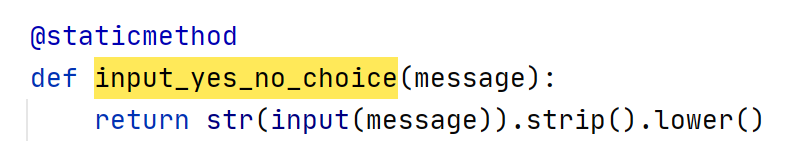
Figure .1 Static Method - print\_current\_tasks\_in\_list



1. input\_yes\_no\_choice(message)

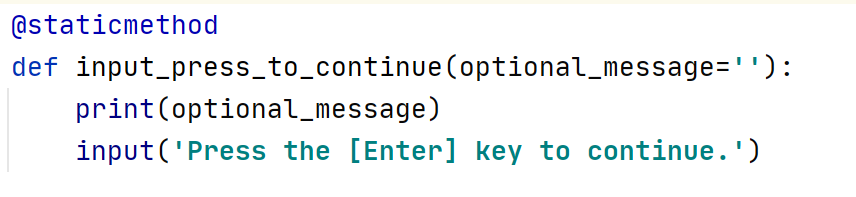
This method takes a message and returns just the user input (should be y or n but could be anything).

Figure .1 Static Method - input\_yes\_no\_choice



1. input\_press\_to\_continue(optional\_message='')  
   This method takes an optional message and displays it to the user and then requests that the user press Enter to continue.

Figure .1 Static Method - input\_press\_to\_continue



## Program Script (Main Body)

The main body of the program has been reduced substantially due to setting up the classes and methods. It reads easily and calls the class methods defined earlier. I will identify the class, method, arguments and return values for each of the code scripts in the program’s main body.

1. Load Data from ToDoFile.txt into lstTable.

strFileName was populated in Global Variables as “ToDoFile.txt”

Reads Data from the “ToDoFile.txt” file to lstTable *Processor.read\_data\_from\_file(strFileName, lstTable)* *# read file data into lstTable*

*Class: Processor*

*Method: read\_data\_from\_file*

*Arguments: strFileName, lstTable*

*Return: lstTable*

1. While Loop & Conditional Statement

While the user has not requested to exit the program (Option 5 currently), the program will continue. These are the steps within the While Loop:

* 1. Prior to the conditional statement options, the current data should be displayed to the screen, then the menu should be displayed to the screen, then the user’s selected menu option need to be captured.
     1. Displays Home Inventory Item Names and Item Values from the list (lstTable) to the screen

**IO.print\_current\_tasks\_in\_list(lstTable)**

*Class: IO*

*Method: print\_current\_tasks\_in\_list*

*Arguments: lstTable*

* + 1. Displays the Menu to the screen

**IO.print\_menu\_tasks()**

*Class: IO*

*Method: print\_menu\_tasks*

* + 1. Request User Menu Choice

**strChoice = IO.input\_menu\_choice**

*Class: IO*

*Method: input\_menu\_choice*

*Returns: strChoice*

* 1. Conditional Statement Option – User chose 1 - Add a New Item
     1. Request New Item and New Value from User

**strTask, strPriority = IO.input\_new\_task\_and\_priority()**

*Class: IO*

*Method: input\_new\_task\_and\_priority()*

*Returns: strTask (New Item) and strPriority (New Value)*

* + 1. Add New Item and New Value to List

**lstTable, strStatus = Processor.add\_data\_to\_list(strTask, strPriority, lstTable)**

*Class: Processor*

*Method: add\_data\_to\_list*

*Arguments: strTask, strPriority, lstTable*

*Returns: lstTable, Success*

* + 1. Ask User to Press Enter to Continue

**IO.input\_press\_to\_continue(strStatus)**

*Class: IO*

*Method: input\_press\_to\_continue*

*Arguments: strStatus (from ii)*

*Displays: Contents of strStatus (probably Success)*

* 1. Conditional Statement Option – User chose 2 – Remove an Existing Item
     1. Ask User for Item to Remove

**strTask = IO.input\_task\_to\_remove()***Class: IO*

*Method: input\_task\_to\_remove*

*Returns: strTask (name of item to remove)*

* + 1. Remove Item from List

**lstTable, strStatus = Processor.remove\_data\_from\_list(strTask,lstTable)**

*Class: IO*

*Method: input\_task\_to\_remove*

*Arguments: strTask, lstTable*

*Returns: lstTable, strStatus*

* + 1. Ask User to Press Enter to Continue

**IO.input\_press\_to\_continue(strStatus)**

*Class: IO*

*Method: input\_press\_to\_continue*

*Arguments: strStatus (from ii)*

*Displays: Contents of strStatus (probably Success)*

* 1. Conditional Statement Option – User chose 3 – Save data to File
     1. Ask user again if they wish to save the data (in lstTable) to the file

**strChoice = IO.input\_yes\_no\_choice("Save this data to file? (y/n) - ")**

*Class: IO*

*Method: input\_yes\_no\_choice*

*Arguments: “Save this data to file? (y/n) –“*

*Returns: strChoice (y or anything else)*

* + 1. If strChoice = y or Y, then write the data to the file

and Ask User to Press Enter to Continue

**if strChoice.lower() == "y":  
 lstTable, strStatus = Processor.write\_data\_to\_file(strFileName,lstTable)**

*Class: Processor*

*Method: write\_data\_to\_file*

*Arguments: strFileName, lstTable*

*Returns: lstTable, strStatus (probably Success)*

**IO.input\_press\_to\_continue(strStatus)**

*Class: IO*

*Method: input\_press\_to\_continue*

*Arguments: strStatus*

*Displays: Contents of strStatus(probably Success)*

* + 1. If strChoice is not y, let user know the data was not saved

and Ask User to Press Enter to Continue  
**else:  
 IO.input\_press\_to\_continue("Save Cancelled!")**

*Class: IO*

*Method: input\_press\_to\_continue*

*Arguments: “Save Cancelled”*

*Displays: Contents of strStatus (probably Success)*

* 1. Conditional Statement Option – User chose 4 – Reload Data to File
     1. Warn User that if they continue, data will be lost

**print("Warning: Unsaved Data Will Be Lost!")**

* + 1. Ask User if they want to reload the data from the file to be sure

**strChoice = IO.input\_yes\_no\_choice("Are you sure you want to reload data from file? (y/n) - ")**

*Class: IO*

*Method: input\_yes\_no\_choice*

*Arguments: “Are you sure you want to reload data from file? (y/n) – “*

*Displays: Contents of strStatus (probably Success)*

*Returns: strChoice (y or anything else)*

* + 1. If strChoice = y or Y, then write the data to the file

and Ask User to Press Enter to Continue

**if strChoice.lower() == 'y':**

**lstTable, strStatus = Processor.read\_data\_from\_file(strFileName, lstTable)  
 IO.input\_press\_to\_continue(strStatus)***Class: Processor*

*Method: write\_data\_to\_file*

*Arguments: strFileName, lstTable*

*Returns: lstTable, strStatus (probably Success)*

**IO.input\_press\_to\_continue(strStatus)**

*Class: IO*

*Method: input\_press\_to\_continue*

*Arguments: strStatus*

*Displays: Contents of strStatus(probably Success)*

* + 1. If strChoice is not y, let user know the reload was cancelled

and Ask User to Press Enter to Continue  
**else:  
 IO.input\_press\_to\_continue("File Reload Cancelled!")**

*Class: IO*

*Method: input\_press\_to\_continue*

*Arguments: “File Reload Cancelled”*

*Displays: Contents of strStatus (probably Success)*

* 1. Conditional Statement Option – User chose 5 – Exit Program
     1. Display Goodbye to the user

And break out of the loop to exit the program

**print("Goodbye!")  
break**

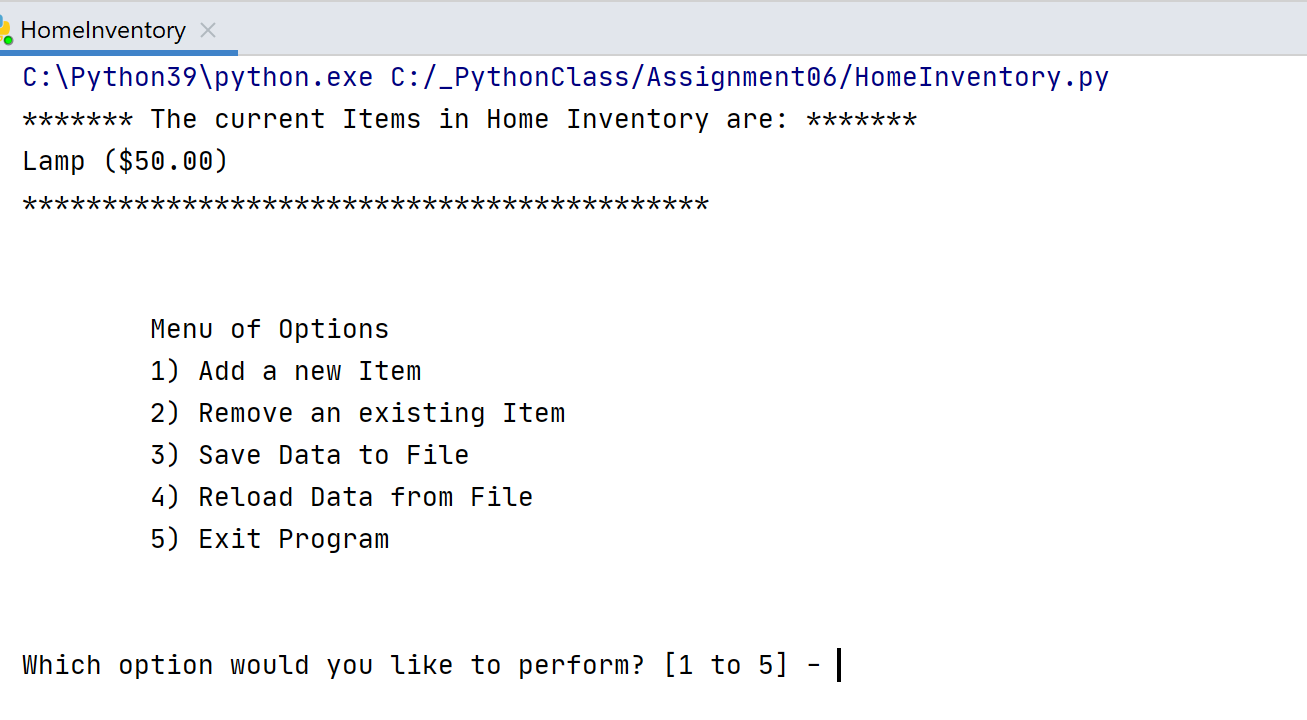
# Testing

After saving the program code to HomeInventory.py, I need to test that the program works as per the requirements.

#### **Test 1 – PyCharm Test**

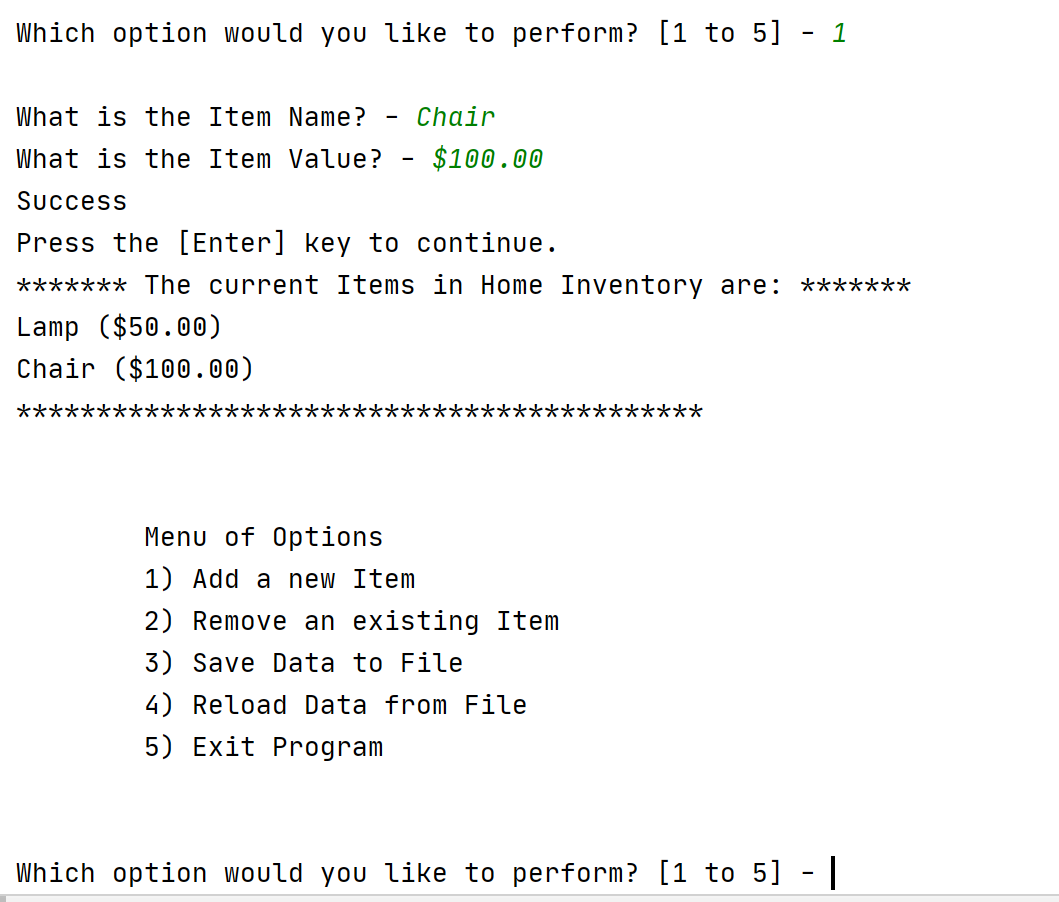
I choose Run in PyCharm and the program provides:

1. A list of items saved from the ToDoFile.txt is displayed, The Menu of Options appears, and a request to select an option is shown.

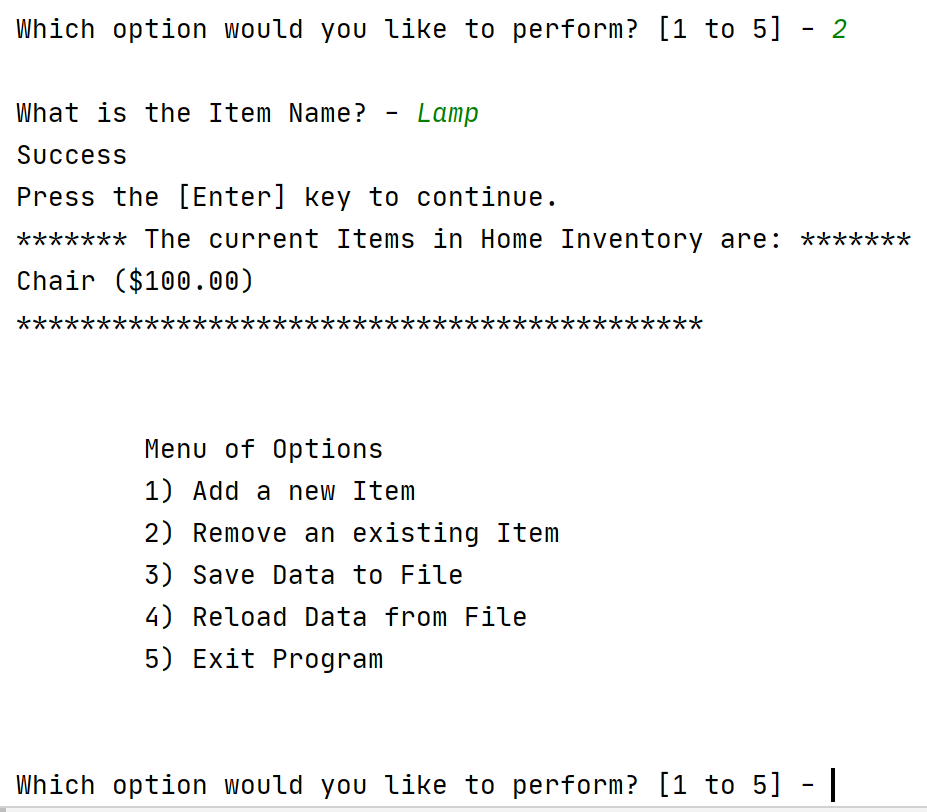


1. When Option 1 is selected, the User is prompted for an Item and a Value.

Selecting Option 1 after shows that the Vase is appended to lstTable.

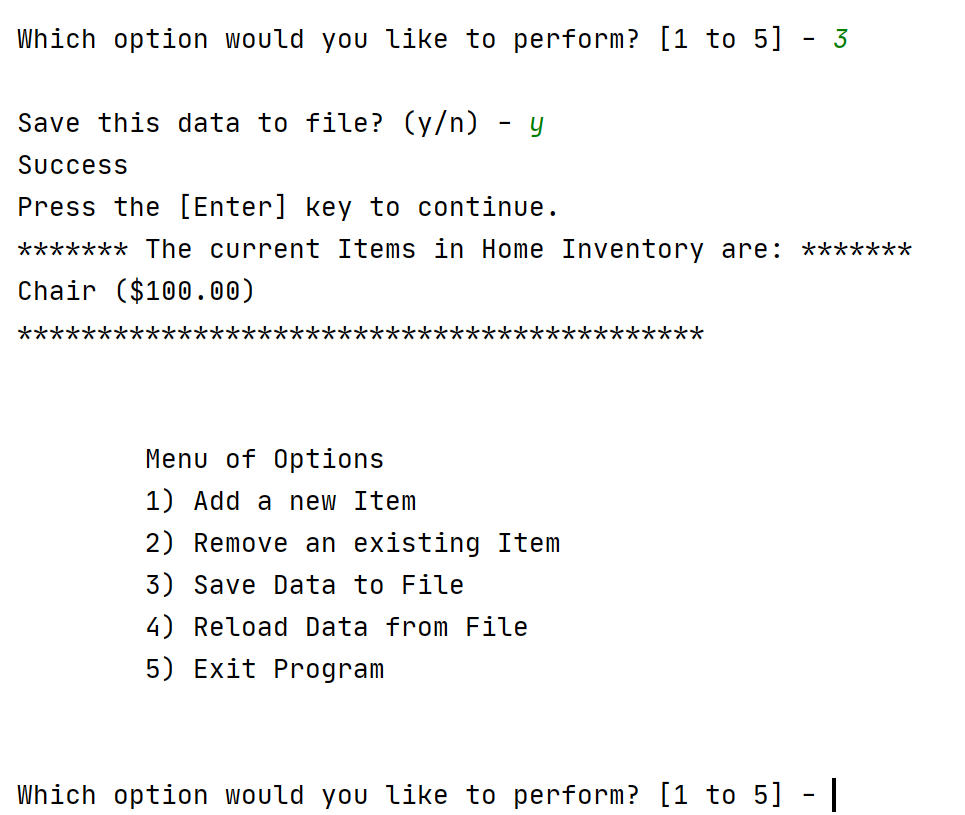


1. When Option 2 is selected, the User is prompted for an Item to Remove (Lamp). This item is found in the lstTable and removed. Lamp used to be the first item.

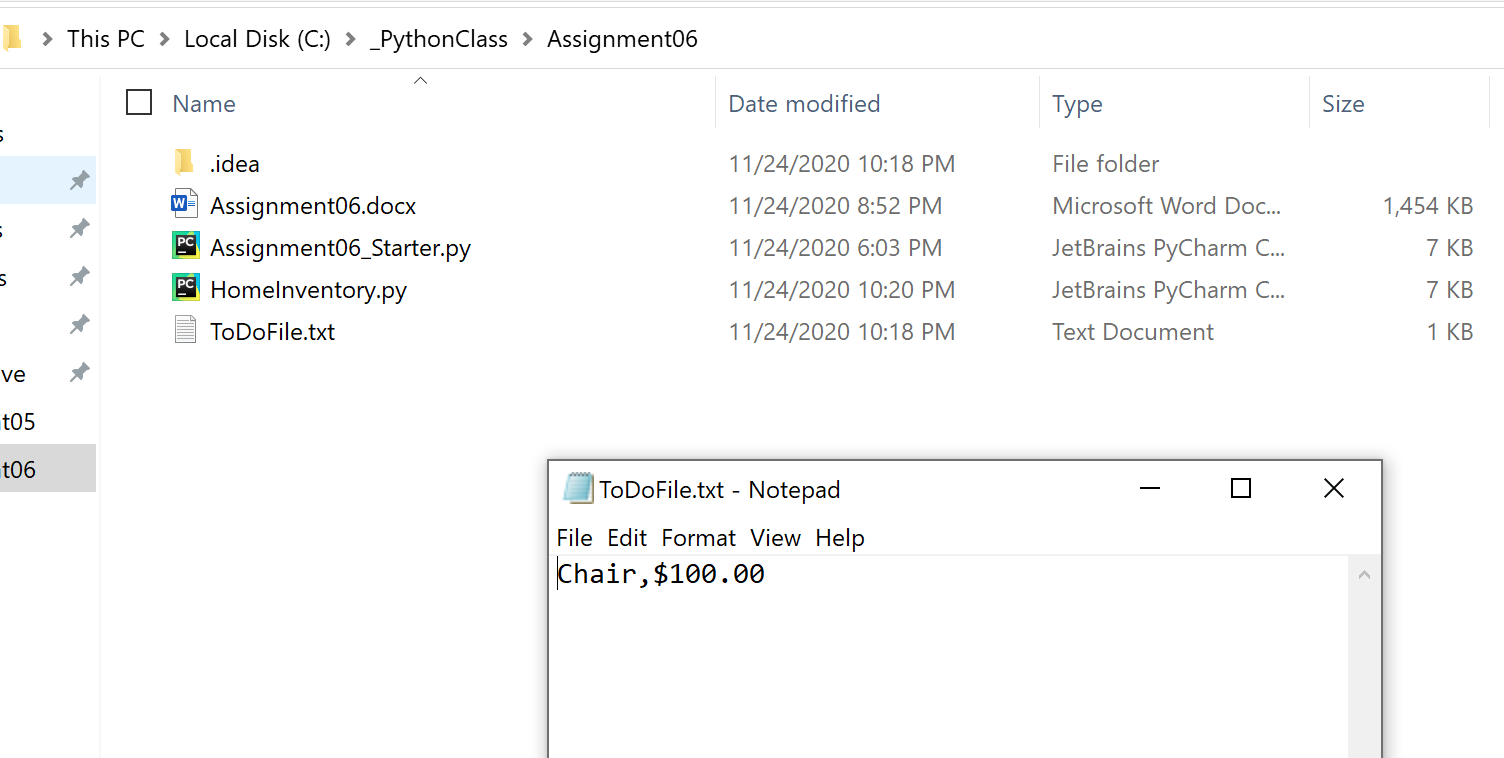


1. When Option 3 is selected:
   1. The user is asked if they want to save the data to the file (y/n)

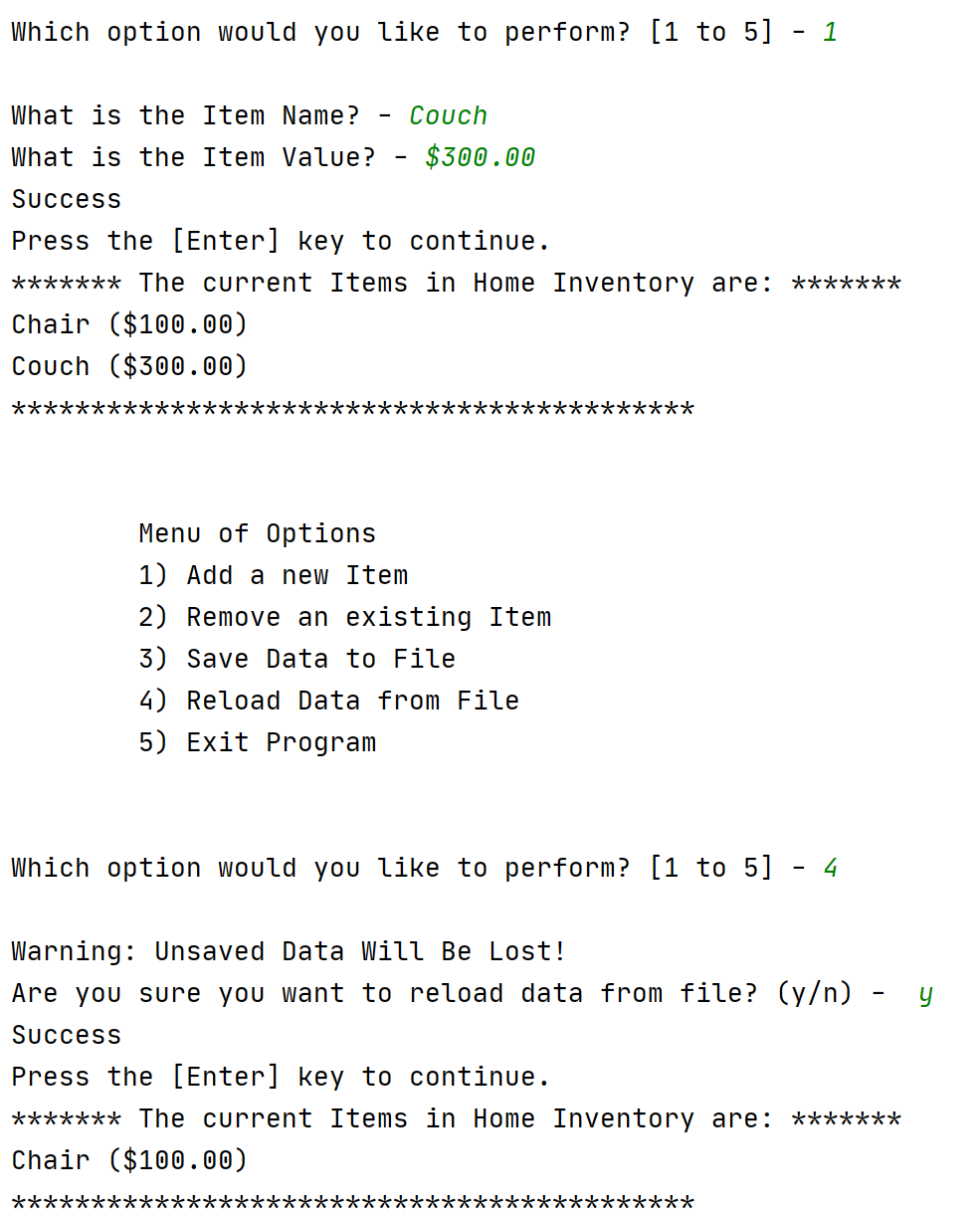
If the user selects ‘y’, the file is saved.



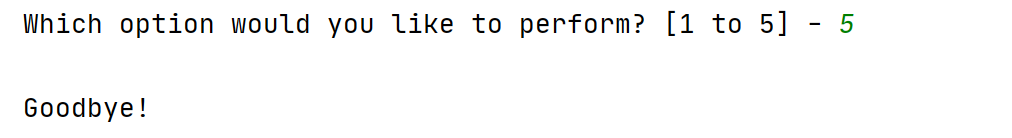
* 1. The data from the lstTable is saved to the ToDoFile.txt file in the same location as the HomeInventory.py. This is the same as a.



1. When Option 4 is selected, the Program displays the message “Warning: Unsaved Data will be Lost!”. The user is then prompted to confirm their choice: “Are you sure you want to reload data from file? (y/n) – “. If the user responds with a “y”, then the data in the file is reloaded and the lstTable is overwritten.



1. When Option 5 is selected, the Program displays “Goodbye!” and Exits.

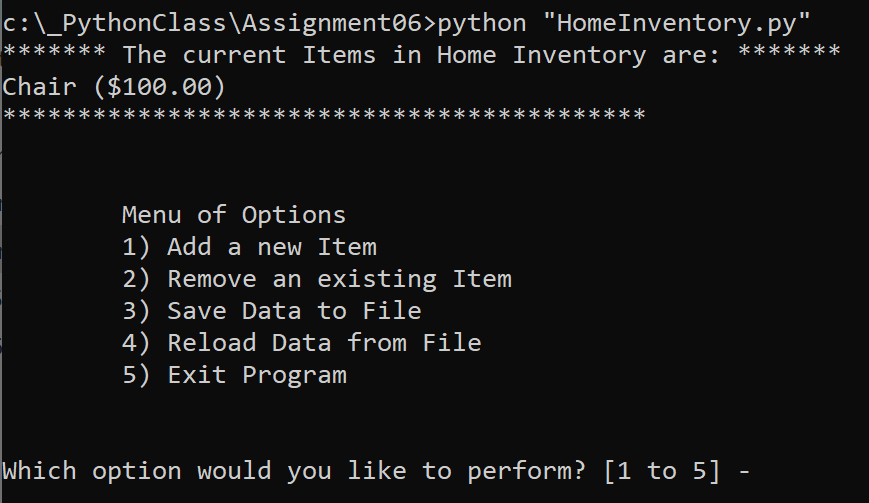


The Home Inventory program requirements have been met for PyCharm.

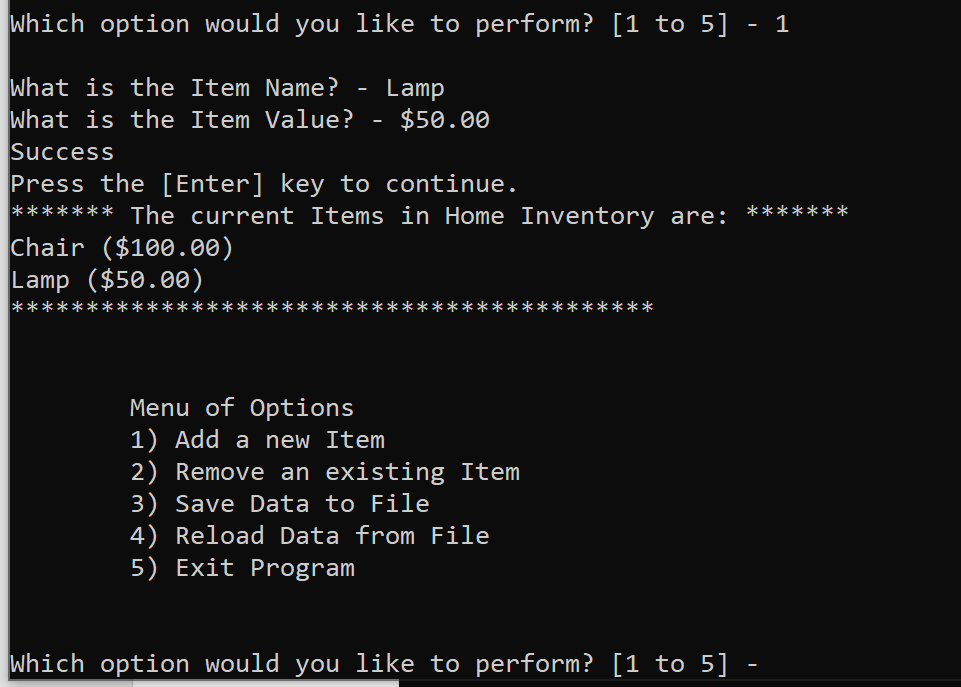
**Test 2 – OS Command Test**

I start the program in the Command Shell and the program provides:

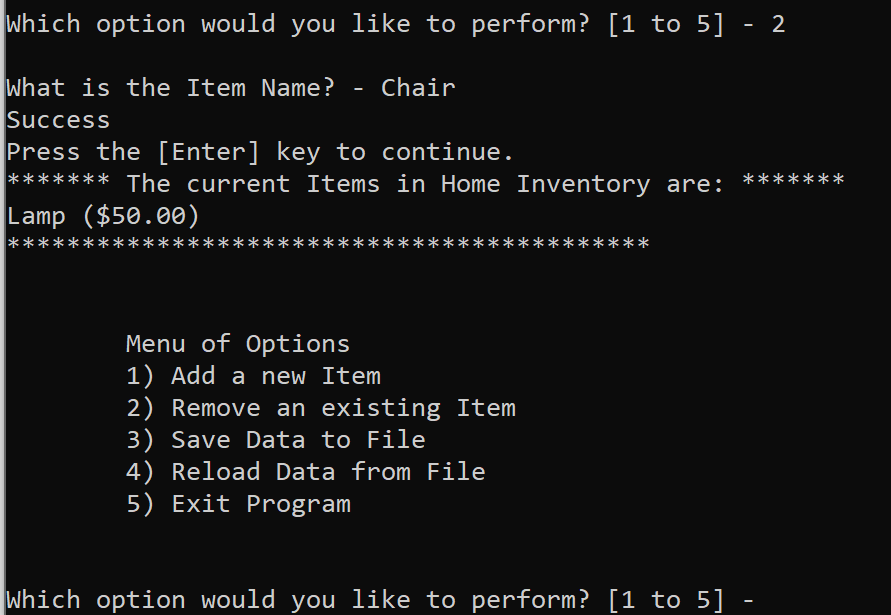
1. A list of items saved from the ToDoList.txt is displayed, The Menu of Options appears, and a request to select an option is shown.



1. When Option 1 is selected, the User is prompted for an Item and a Value.

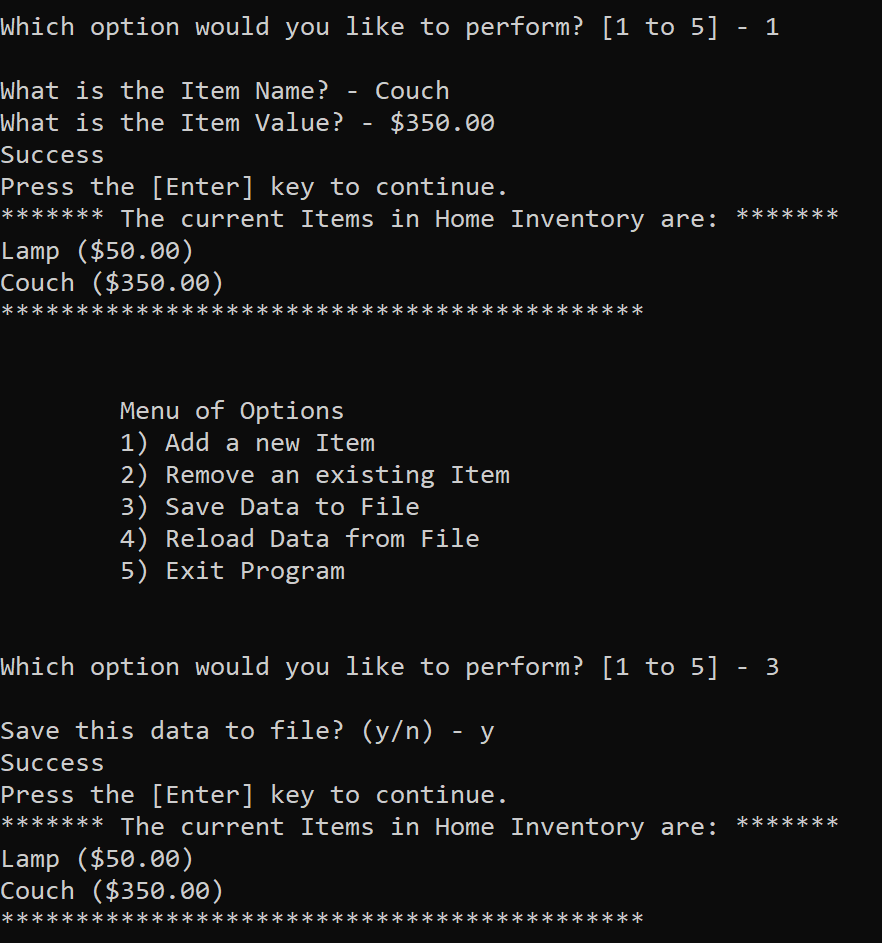


1. When Option 2 is selected, the User is prompted for an Item to Remove (Chair). This item is found in the lstTable and removed. Chair used to be at the top of the list.

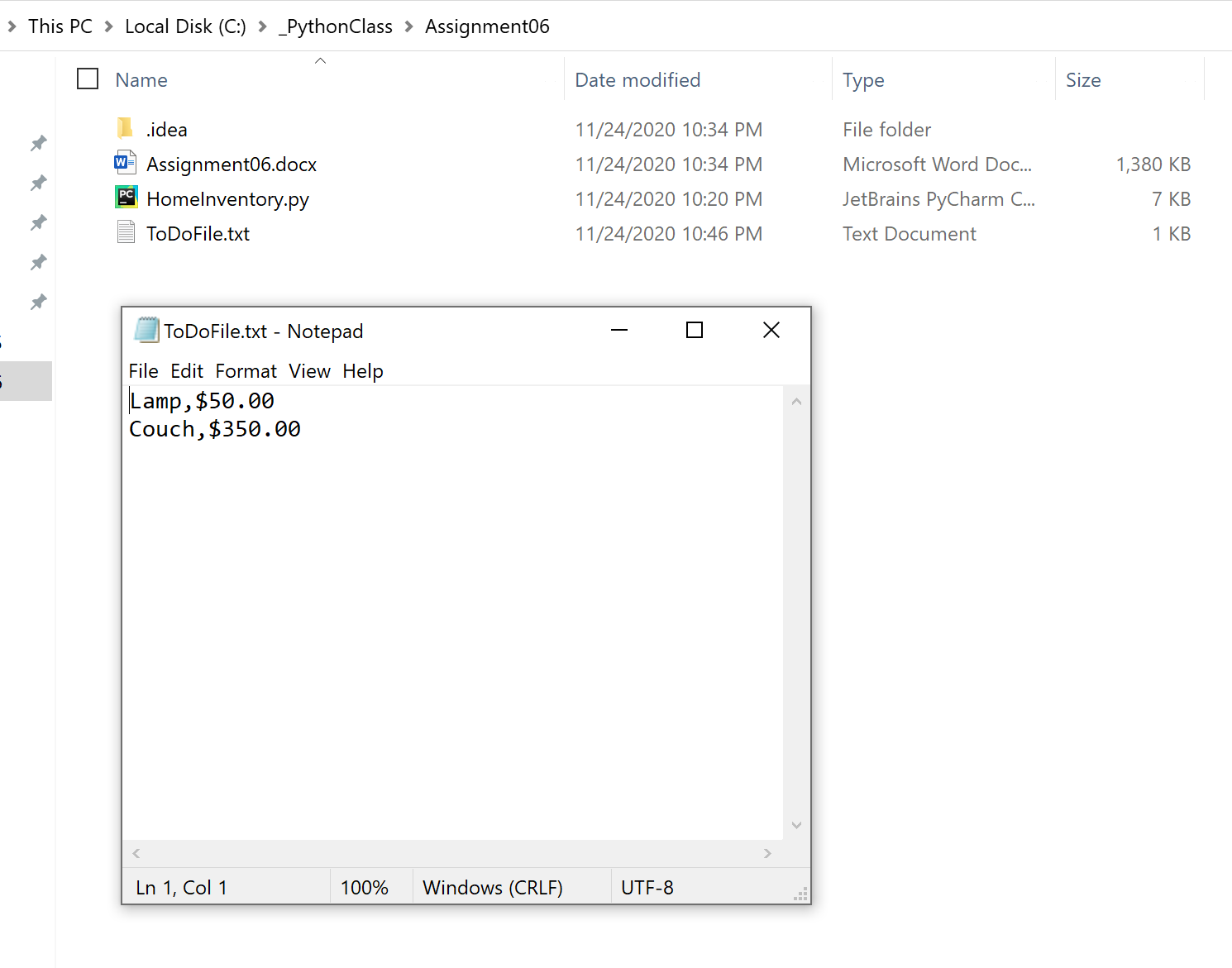


1. When Option 3 is selected:
   1. The user is asked if they want to save the data to the file (y/n)

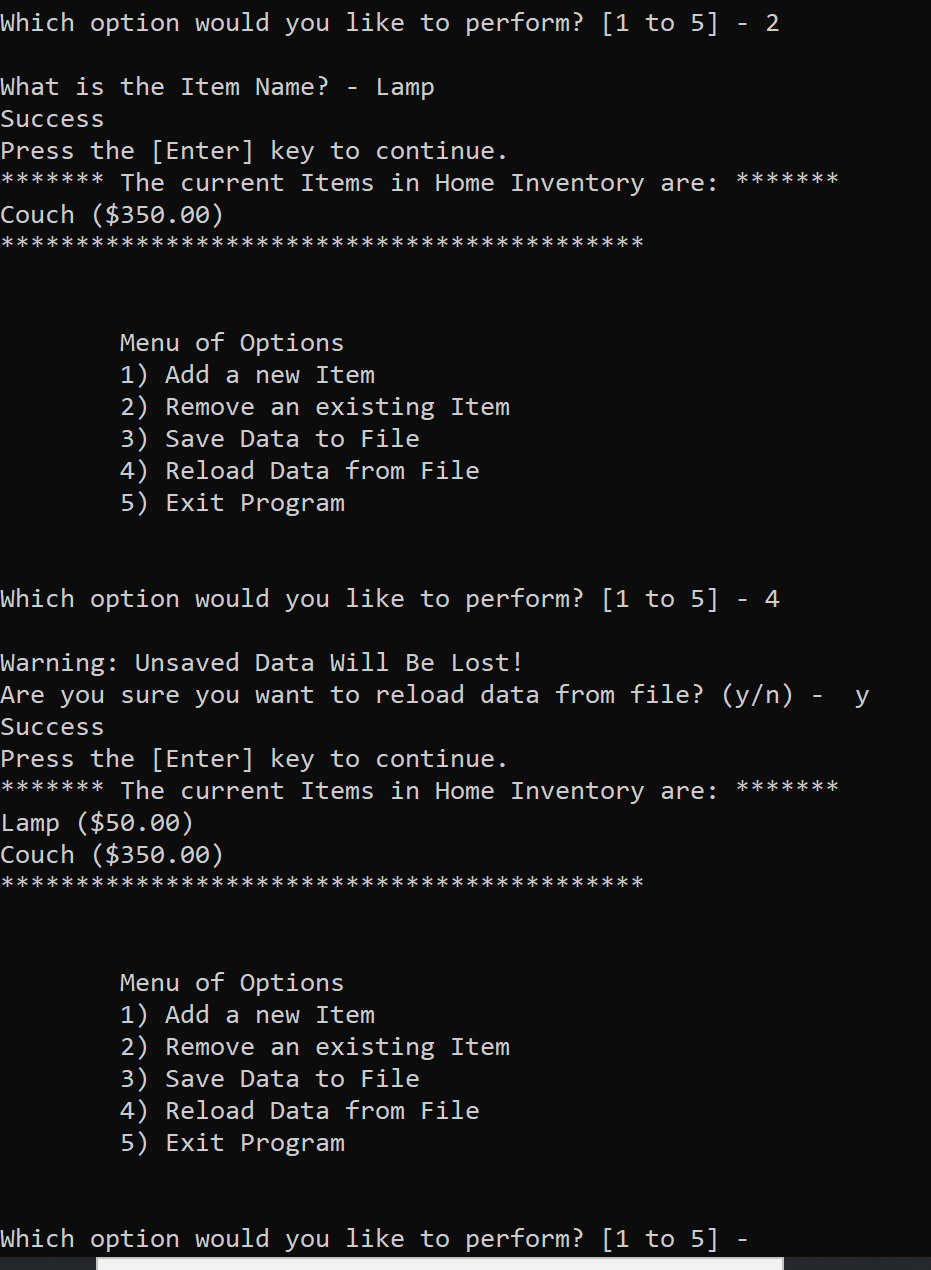
If the user selects ‘y’, the file is saved.



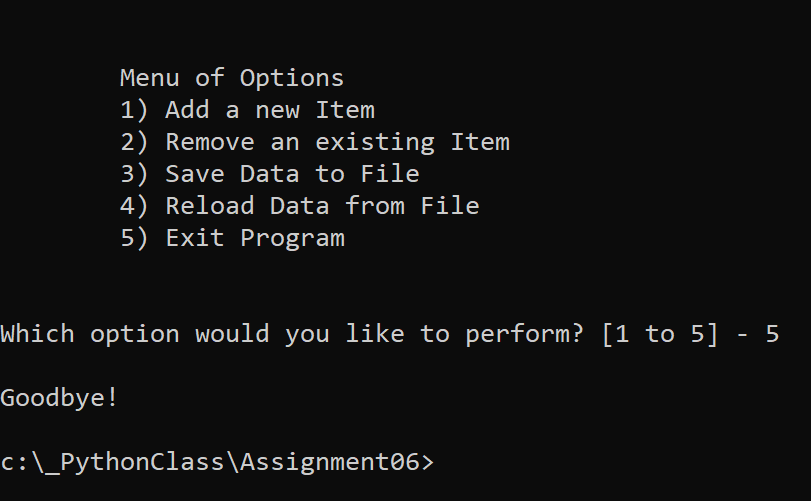
* 1. The data from the lstTable is saved to the ToDoFile.txt file in the same location as the HomeInventory.py.



1. When Option 4 is selected, the Program displays the message “Warning: Unsaved Data will be Lost!”. The user is then prompted to confirm their choice: “Are you sure you want to reload data from file? (y/n) – “. If the user responds with a “y”, then the data in the file is reloaded and the lstTable is overwritten.



1. When Option 5 is selected, the Program displays “Goodbye!” and Exits.



The Home Inventory program requirements appear to have been met for OS Command as well.

# Conclusion

Using classes and static methods with the Program Script (Main Body) was very helpful to separate concerns within the program. It was interesting to see a static method could be used several times within the program code – for instance the IO.input\_press\_to\_continue method. The argument being passed could be different each time. It became clear that program code could be reused easily by implementing classes and methods. Finally, once again, sharing code with other students in GitHub as well as reviewing other’s code is helpful to learn alternative coding practices.