Luis Valderrama

08/14/2021

IT FDN 110 A

Module 07

[GitHub Repository Module07](https://github.com/lvalderr/-IntroToProg-Python-Mod07)

[GitHub Blog Module07](https://lvalderr.github.io/-IntroToProg-Python-Mod07/)

Files and Exceptions

# Intro

During the seventh module, I learned the concepts of working with files and exceptions. This document covers the steps to create a script named, *Assignment07.py*, designed to prompt the user to select from a seven-option menu and execute the program based on the selection. This script introduces the concept of Try/Except error handling and “pickling” concepts. The script is created using PyCharm as the Integrated Development Environment (IDE) and runs in PyCharm and Command Prompt.

# Assignment06\_Starter.py Program Set UP

In this section we look at the initial set up of the program script.

## Script Header and Objective

As mentioned in prior documents, the first part of the script outlines not only the title, description, date, and change log. But it also provides the reader with a high-level objective of the script and what it is designed to do. (Figure 1.1)

# --------------------------------------------------------------------------------------------- #  
# Title: Assignment 07  
# Description: The script demonstrates the use of Try - Exception handling in a Menu  
# LValderrama, 8.09.2021, Created started script  
# LValderrama, 8.10.2021, Added Try/Catch to Menu, read from file, write to file, entering inventory values  
# --------------------------------------------------------------------------------------------- #  
  
# Objective:  
# 1. Present a menu of choices for the user to select from.  
# 1) Display current inventory  
# 2) Add a new inventory item  
# 3) Remove an existing inventory item  
# 4) Save data to file  
# 5) Reload data from text file  
# 6) Reload data from pickle file  
# 7) Exit program  
# 2. Use Try/Catch Error Handling

# 3. The data is saved in both. a text file and a .dat file

# Improving my code: Entered a Try/Catch to:  
# 1. The menu option selection. the user should only enter integers between 1 - 5.  
# 2. Option 1 Read from file  
# 3. Option 2 Value as the user should only enter integers.  
# 4. Option 4 writing to file in case the file does not exit.

***Figure 1.1: Script Header and Objective***

## Pseudo-Code

Before developing the script, the steps are outlined in the form of Pseudo-Code (Figure 1.2) to help translate the objective into the programing code and develop a usable program. In this example, there are 12 core steps that may expand into sub-steps as the script develops.

# Pseudo-Code:  
# Data ---------------------------------------------------------------------------------------- #  
# Step 1 - Declare variables and constants ---------------------------------------------------- #

# Processing Data ----------------------------------------------------------------------------- #  
# Step 2 - set up class processor functions---------------------------------------------------- #

# Input/Output -------------------------------------------------------------------------------- #  
# Step 3 - set up presentation, IO functions -------------------------------------------------- #

# Main Body of the Script --------------------------------------------------------------------- #  
# Step 4 - When the program starts, load data from BikeInventory.txt

# Step 4 - Display a menu of choices to the user

# Step 5 Calling the function to display menu of options

# Step 6 Calling the function to display current inventory

# Step 7 Calling the function to enter new data to list

# Step 8 Calling the function to remove data from list

# Step 9 Calling the function to save data from list to .txt and "pickling" data to .dat file

# Step 10 Calling the function to reload data from .txt file to list

# Step 11 Calling the function to "unpickle" data from .dat file to list

# Step 12 Ending the program and exit

***Figure 1.2: Pseudo-Code***

## Declaring Variables and Constants

As best practice, the variables and constants are declared before inserting the codes. (Figure 1.3)

# Step 1 - Declare variables and constants ---------------------------------------------------- #  
import sys  
pickle = ""  
pickleFile = "BikeInventoryPickle.dat" # This is the pickle file.  
strFileName = "BikeInventory.txt" # The name of the data file  
objFile = None # An object that represents a file  
dicRow = {} # A row of data separated into elements of a dictionary {Bike, Value}  
lstTable = [] # A list that acts as a 'table' of rows  
strChoice = "" # Captures the user option selection  
strBike = "" # Captures the user inventory data

***Figure 1.3: Declaring Variables and Constants***

## Class Processor

This assignment uses six processing functions as outlined below in (Figure 1.4). Each function will be used for a specific process in the Menu of Options and will be called based upon the user’s input. I set up a Class Processor to group the functions, variables and constants using @staticmethod to define the utility methods or group logically related functions into the class. Then, I insert the variables, parameters for each function. Each function will be examined in more detail as we start looking at section 2.0 Menu of Options.

Text

Description automatically generated

***Figure 1.4: Class Processor***

## Class IO (Input/Outputs)

This assignment uses six additional functions as outlined below in (Figure 1.5). Each function will be used for a specific presentation process in the Menu of Options and will be called based on the user’s input. A Class IO (input/output) is set up to group the functions, variables and constants using @staticmethod. These functions are intended to take input from the user and display prompts and results. Then, I insert the variables for each function. Each function will be examined in more detail as we start looking at section 2.0 Menu of Options.

Text

Description automatically generated

***Figure 1.5: Class IO (Inputs/Outputs)***

## Loading Data from Text File to the Table

When the program starts it opens the BikeInventory.txt and loads the two-column content into a list held in a table named **lstTable** as shown in (Figure 1.6.a).

The function is defined using **def** followed by the name of the function, in this case **read\_data\_from\_text\_file** and the variable given is **file\_name** which represents the text file BikeInventory.txt where the list of bikes is saved. The parameters are entered as well as a return which is this example is the **lstTable**. The actual function code is entered to open the .txt file, retrieve the data, and place it in the **lstTable** in the form of append. The .txt file is then closed, and a return will display the items in the **lstTable**. The function is encased in a Try/Except error handling to prevent the program to error and crash if the user makes the wrong entry. *“An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.*

*When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.” “If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible”* [*https://www.tutorialspoint.com/python/python\_exceptions*](https://www.tutorialspoint.com/python/python_exceptions)*,* (external site). I find the tutorial linked helpful because it provides definitions, various examples, and exception name & descriptions.

@staticmethod  
def read\_data\_from\_text\_file(file\_name):  
 *""" Reads data from BikeInventory.txt file into a list of dictionary rows* ***:param*** *file\_name: (string) strFileName = BikeInventory.txt* ***:return****: (list) dictionary rows  
 """* try:  
 lstTable = []  
 objFile = open(file\_name, "r")  
 except IOError:  
 print('\nThere is no file associated with this application. ')  
 input("\nPress Enter to exit.")  
 sys.exit()  
 for row in objFile:  
 lstRow = row.split(",")  
 dicRow = {"Bike": lstRow[0].strip(), "Value": lstRow[1].strip()}  
 lstTable.append(dicRow)  
 objFile.close()  
 return lstTable

***Figure 1.6.a: Class Processor. read\_data\_from\_text\_file***

The **Class Processor. read\_data\_from\_text\_file** functionis called outside of the While loop containing the Menu of Options. (Figure 1.6.b.). Additionally, the variable **strFileName** represents the BikeInventory.txt (Figure 1.6.c) where the information is held and will be loaded to the list and displayed on the screen when the user enters option 1.

lstTable = Processor.read\_data\_from\_text\_file(strFileName)

***Figure 1.6.b: Calling the function. read\_data\_from\_text\_file***

Graphical user interface, text

Description automatically generated

***Figure 1.6.c: View of BikeInventory.txt content***

# Menu of Options Set Up

In this assignment, the program initiates with a menu that offers seven choices as shown below in (Figure 2.0.a). The menu is defined in the **Class IO** (input/output) as **print\_menu\_of\_options**, without variables. (Figure 2.0.a)

@staticmethod  
def print\_menu\_of\_Options():  
 *""" Display a menu of choices to the user* ***:return****: Nothing  
 """* print('''  
 Menu of Options  
 1) Display current inventory  
 2) Add a new inventory item  
 3) Remove an existing inventory item  
 4) Save data to file  
 5) Reload data from text file  
 6) Reload data from pickle file  
 7) Exit program  
 ''')  
 print() # Adding extra line

***Figure 2.0.a: Class IO. print\_menu\_of \_options***

An **input()** function is defined as **IO**.**input\_menu\_choice()** in the form of an integer. This input is established to get a choice from 1 – 7 from the user (Figure 2.0.b). The choice will then activate the corresponding function and the program will process the data accordingly. Since the Menu of Options is encased in a Try/Except error handling, if the user enters a non-numerical value the program will display an error message and request the user to enter a numerical value to proceed.

@staticmethod  
def input\_menu\_choice():  
 *""" Gets the menu choice from a user* ***:return****: string  
 """* choice = int(input("Which option would you like to perform? [1 to 7] - "))  
 print() # Add an extra line for looks  
 return choice

***Figure 2.0.b: Class IO. input\_menu\_choice***

Another **input()** function is defined as **IO.input\_press\_to\_continue**. This input is established for the sole purpose of pausing the program before advancing to the next step and allow the user to control when the program should continue. (Figure 2.0.c.)

@staticmethod  
def input\_press\_to\_continue(optional\_message=''):  
 *""" Pause program and show a message before continuing* ***:param*** *optional\_message: An optional message to display* ***:return****: Nothing  
 """* print(optional\_message)  
 input('Press the [Enter] key to continue.')

***Figure 2.0.c: Class IO. input\_press\_to\_continue***

The functions previously described are called at the beginning of the **while(True)** loop. The program will continue to loop through until a break is reached. (Figure 2.0.d.).

while True:  
 try:  
  
# Step 5 Calling the function to display menu of options  
 IO.print\_menu\_of\_Options()  
 strChoice = IO.input\_menu\_choice()

***Figure 2.0.d: Calling the function to display menu of options and enter a choice***

Displayed below in (Figure2.0.e.) is the program as it starts running in PyCharm. The user can also double click on the .py file itself and the program will start running in Command Prompt by default. The image below shows the Menu of Options 1 - 7 which is the result of calling the function **IO.print\_menu\_choice()** while the **input()** identified as “Which option would you like to perform? [1 to 7] is the result of calling the function **strChoice = IO.input\_menu\_choice().**

Text

Description automatically generated

***Figure 2.0.e: Menu of Options shown in PyCharm and Command Prompt as the program starts running***

## Display Current Inventory (Menu Option 1)

In this section we examine the programing to execute option 1 from the Menu of Options (Figure 2.1.a)

There is no Processing function. However, there is an Input/Output function that is defined as **print\_current\_inventory\_in\_list** with the variable, **list\_of\_rows,** which is used to hold the data in the form of a two-column list. The function prints the data in the form of a list of bikes and values. The **print()** is set up to “unpack” the data.

@staticmethod  
def print\_current\_inventory\_in\_list(list\_of\_rows):  
 *""" Shows the current bikes in the list of dictionaries rows* ***:param*** *list\_of\_rows: (list) split data into list* ***:return****: Nothing  
 """* print("\*\*\*\*\*\*\* The current inventory: \*\*\*\*\*\*\*")  
 for row in lstTable: # Displaying rows below in the form of a vertical list  
 print(row["Bike"] + ',' + row["Value"]) # Unpacking  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print() # Add an extra line for looks

***Figure 2.1.a.: Class IO.print\_current\_inventory\_in\_lists (Menu Option 1)***

The function previously described is called within the **while(True)** loop when the user enters “1” in the **input\_menu\_choice** as prompted by the message “Which option would you like to perform? [1 – 7]”. The next function called is **input\_press\_to\_continue()** which is intended to pause the program until the user presses Enter to proceed. (Figure 2.1.b.)

if strChoice == 1:  
 IO.print\_current\_inventory\_in\_list(lstTable)  
 IO.input\_press\_to\_continue()  
 continue

***Figure 2.1.b.: Calling the function (Menu Option 1)***

Displayed below in (Figure2.1.c.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.1.c.: Results from Menu Option 1 displayed in PyCharm and Command Prompt***

## Add a New Inventory Item (Menu Option 2)

In this section we examine the programing to execute option 2 from the Menu of Options (Figure 2.2.a.)

The processing function is defined as **add\_data\_to\_list** with the variables, **list\_of\_rows, bike, value** in the form of strings. The function itself takes the bike and value entered by the user and appends the lstTable that holds the data with the new items. The function **returns** a value in the form of an expression immediately after the user enters the new data. The value is the list of bikes and values.

@staticmethod  
def add\_data\_to\_list(list\_of\_rows, bike, value):  
 *""" Adds bikes and values to list* ***:type*** *list\_of\_rows:* ***:param*** *list\_of\_rows: (list) you want filled with file data* ***:param*** *bike: (string) to do item* ***:param*** *value: (string) the value of the bike* ***:return****: (list) of bikes and values  
 """* try:  
 dicRow = {"Bike": str(bike), "Value": str(value)}  
 list\_of\_rows.append(dicRow)  
 except ValueError:  
 print("Enter a number in Value")  
 return list\_of\_rows, 'Success'

***Figure 2.2.a.: Class Processor.add\_data\_to\_list (Menu Option 2)***

The Input/Output function is defined as **input\_new\_inventory\_and\_value** without parameters. The function gets two inputs, bike and value from the user, and the entries are encased in a Try/Except error handling that will return an error message indicating to the user that an integer value must be entered for the value. (Figure 2.2.b)

@staticmethod  
def input\_new\_inventory\_and\_value():  
 *""" Adds new bike and value entered by the user* ***:return****: (string) Bike and its corresponding value  
 """* try:  
 bike = str(input("Enter a new bike inventory: ")).strip()  
 value = int(input("Enter value: "))  
 print()  
 return bike, value  
 except ValueError:  
 print("Enter a number in Value")

***Figure 2.2.b: IO.input\_new\_inventory\_and\_value (Menu Option 2)***

The functions previously described are called within the **while(True)** loop when the user enters “2” in the **input\_menu\_choice** as prompted by the message “Which option would you like to perform? [1 – 7]”. The next function called is **IO.input\_new\_inventory\_and\_value()** which gets the data entered by the user, then the next function called is **Processor.add\_data\_to\_list** which appends the **lstTable** with the new bike and its value. The program calls **IO.Print\_current\_inventory\_in\_list** function which now displays the list updated with the new bike and value entered by the user. The last function called is **IO.input\_press\_to\_continue()** in order to pause the program until the user presses Enter to proceed. (Figure 2.2.c.)

elif strChoice == 2:  
 tplData = IO.input\_new\_inventory\_and\_value()  
 Processor.add\_data\_to\_list(lstTable, tplData[0], tplData[1])  
 IO.print\_current\_inventory\_in\_list(lstTable)  
 IO.input\_press\_to\_continue()  
 continue

***Figure 2.2.c.: Calling the function (Menu Option 2)***

Displayed below in (Figure2.2.d.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.2.d.: Results from Menu Option 2 displayed in PyCharm and Command Prompt***

## Remove an Existing Inventory Item (Menu Option 3)

In this section we examine the programing to execute option 3 from the Menu of Options. (Figure 2.3.a.)

The processing function is defined as **remove\_data\_from\_list** with the variable, **list\_of\_rows,** in the form of string. The function itself gets an input from the user and compares to the list of bikes and values, **lstTable**. If the input is matched with an item in the list, then the function removes the bike and value from the list and displays a message confirming the item has been removed. If the input is not matched to the list, then the program prints a message indicating the item is not found.

@staticmethod  
def remove\_data\_from\_list(list\_of\_rows):  
 *""" Removes bikes from list* ***:param*** *list\_of\_rows: (list) you want filled with file data* ***:return****" (List) of bikes removed  
 """* while len(lstTable) > 0:  
 print("Current inventory list:\n", "\n".join([f"\t{item['Bike']}" for item in lstTable]))  
 term = input("Enter the Bike you want to delete or type 'exit' to return to Menu of Options: ")  
 if term.lower() == "exit":  
 break  
 for bike in lstTable:  
 if term in bike["Bike"]: # If the bike entered by the user matches the name in the list then proceed  
 print(f"Removing {bike['Bike']}...") # Print a message "item found" and processing removal  
 list\_of\_rows.remove(bike) # The item is removed from the table  
 break  
 else:  
 print(f"Bike not found in list: {term}") # Displays message indicating the bike was not found!

***Figure 2.3.a.: Class Processor.remove\_data\_from\_list (Menu Option 3)***

The Input/Output function is defined as **remove\_data\_from\_list** with parameter **term** (Figure 2.3.b). The function gets one input from the user, and the processing function previously mentioned will run through the matching, and removal process.

@staticmethod  
def remove\_data\_from\_list(term):  
 *""" Removes bikes and values entered by the user  
  
 :term: (string) this is the item the user wants to remove* ***:return****: (string) Bike and its corresponding value  
 """* try:  
 term: input("Confirm the bike you want to delete or type 'exit' to return to Menu of Options: ")  
 except ValueError:  
 print('Invalid input')

***Figure 2.3.b.: IO.remove\_data\_from\_list (Menu Option 3)***

The functions previously described are called within the **while(True)** loop when the user enters “3” in the **Processor.remove\_data\_to\_list** which removes item entered by the user from the **lstTable**. The last function called is **IO.input\_press\_to\_continue()** in order to pause the program until the user presses Enter to proceed. (Figure 2.3.c.)

elif strChoice == 3:  
 Processor.remove\_data\_from\_list(lstTable)  
 IO.input\_press\_to\_continue()  
 continue

***Figure 2.3.c.: Calling the function (Menu Option 3)***

Displayed below in (Figure2.3.d.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.3.d.: Results from Menu Option 3 displayed in PyCharm and Command Prompt***

## Saving Data in .txt File (Menu Option 4)

In this section we examine the programing to execute option 4 from the Menu of Options (Figure 2.4.a)

The processing function is defined as **write\_data\_from\_list\_to\_text\_file** with the variable, **file\_name** and **list\_of\_rows** in the form of string. The function itself opens the BikeInventory.txt and saves the data from the bikes and values list. The program then closes the .txt file and prints the message with confirmation the data has been saved in the BikeInventory.txt.

@staticmethod  
def write\_data\_from\_list\_to\_text\_file(file\_name, list\_of\_rows):  
 *""" Writes data from list to text file* ***:param*** *file\_name: (string) strFileName = BikeInventory.txt* ***:param*** *list\_of\_rows: (list) you want filled with file data  
 """* validFile = False  
 while not validFile:  
 try:  
 objFile = open(file\_name, 'w')  
 print()  
 validFile = True  
 except IOError as e:  
 print("Unable to open the file", file\_name, "Ending program.\n", e)  
 input("\nPress Enter to exit.")  
 sys.exit()  
 for row in list\_of\_rows:  
 objFile.write(row["Bike"] + "," + str(row["Value"]) + "\n")  
 objFile.close()

***Figure 2.4.a: Class Processor.write\_data\_from\_list\_to\_text\_file (Option 4)***

The function previously described is called within the **while(True)** loop when the user enters “4” in the **Processor.write\_data\_from\_list\_to\_text\_file**. The program also calls the **IO.print\_current\_inventory\_in\_list** function which displays the current bikes and values. Alternatively, the program saves the data to a .dat file which is also known as “pickling” data. (Figure 2.4.b.) “*The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” 1 or “flattening”.* *<https://docs.python.org/3/library/pickle.html>* (external site). I find the linked helpful because it provides a simple explanation of what pickling is and what it does.

elif strChoice == 4:  
 print('\n Would you like to save your data?')  
 strSaveToFileInput = input("Enter 'y' or 'n': ")  
 if strSaveToFileInput == 'n':  
 print('Data not saved!')  
 if strSaveToFileInput == 'y':  
 Processor.write\_data\_from\_list\_to\_text\_file(strFileName, lstTable)  
 print('\nYour data is saved to', strFileName)  
 Processor.write\_data\_from\_list\_to\_pickle\_file(pickleFile, lstTable)  
 print('Your data is also saved in', pickleFile)  
 IO.print\_current\_inventory\_in\_list(lstTable)  
 IO.input\_press\_to\_continue()  
 continue

***Figure 2.4.b: Calling the function (Option 4)***

Displayed below in (Figure2.4.c.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.4.c: View of the Option 4 as it runs in PyCharm and Command Prompt***

To view the saved data simply open the BikeInventory.txt. (Figure 2.4.d)

Graphical user interface, text

Description automatically generated

***Figure 2.4.d: View of the BikeInventory.txt content***

To view the saved in the BikeInventory.dat, you may open the file with Notepad. (Figure 2.4.e)

A picture containing text, screenshot, monitor, indoor

Description automatically generated

***Figure 2.4.e: Figure 2.4.e: View of the BikeInventoryPickle.dat content***

## Reload Data from .txt File (Menu Option 5)

In this section we examine the programing to execute option 5 from the Menu of Options (Figure 2.5.a)

When the user enters 5 in the **IO**.**input\_menu\_choice()** the program calls the function **lstTable = Processor.read\_data\_from\_file(strFile)** to reload the information from the BikeInventory.txt into the list table. The program then displays the data by way of calling the function **IO.print\_current\_inventory\_in\_list(lstTable)**. The last function called is **IO.input\_press\_to\_continue()** in order to pause the program until the user presses Enter to proceed.

elif strChoice == 5:  
 print("This action will overwrite all unsaved inventory!")  
 strAcceptOrDecline = input("Update data without saving? type 'y' or 'n' ")  
 if strAcceptOrDecline.lower() == 'y':  
 lstTable.clear()  
 lstTable = Processor.read\_data\_from\_text\_file(strFileName)  
 IO.print\_current\_inventory\_in\_list(lstTable)  
 else:  
 input("File no updated. Press Enter to go back to Menu of Options")  
 IO.print\_menu\_of\_Options()  
 continue

***Figure 2.5.a.: Calling the function (Option 5)***

Displayed below in (Figure2.5.b.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

Text

Description automatically generated

***Figure 2.5.b.: Display Option 5 as it runs in PyCharm and Command Prompt***

## Reload Data form .dat file “unpickling” (Menu Option 6)

In this section we examine the programing to execute option 4 from the Menu of Options (Figure 2.6.a)

The function is defined using **def** followed by the name of the function, in this case **read\_data\_from\_pickle\_file** and the variable given is **pickling** which represents the binary file **BikeInventoryPickle.dat** where the list of bikes is saved. The actual function code is entered to open the .dat file, retrieve and display the data.

@staticmethod  
def read\_data\_from\_pickle\_file(pickling):  
 *""" Reads data from BikeInventoryPickle.dat file into a list of dictionary rows* ***:param*** *pickling: (string) pickleFile = BikeInventoryPickle.dat* ***:return****: (list) dictionary rows  
 """* import pickle  
 print('\nUnpickling Bike Inventory.')  
 pickleFile = open(pickling, "rb")  
 pickleFileData = pickle.load(pickleFile) # Error indicating 'str' object has no attribute 'load'  
 print(pickleFileData)  
 pickleFile.close()

***Figure 2.6.a.: Display Option 6 as it runs in PyCharm and Command Prompt***

When the user enters 6 in the **IO.input\_menu\_choice()** the program calls the function **lstTable = Processor.read\_data\_from\_pickle\_file(pickleFile)** to reload the information from the **BikeInventoryPickle.dat** into the list table. The program then displays the data by way of calling the function **IO.print\_current\_inventory\_in\_list(lstTable).** The last function called is **IO.input\_press\_to\_continue()** in order to pause the program until the user presses Enter to proceed.

# Step 11 Calling the function to "unpickle" data from .dat file to list  
 elif strChoice == 6:  
 print("This action will overwrite all unsaved inventory!")  
 strAcceptOrDecline = input("Update data without saving? type 'y' or 'n' ")  
 if strAcceptOrDecline.lower() == 'y':  
 lstTable.clear()  
 lstTable = Processor.read\_data\_from\_pickle\_file(pickleFile)  
  
 else:  
 input("File no updated. Press Enter to go back to Menu of Options")  
 IO.print\_menu\_of\_Options()  
 continue

***Figure 2.6.b.: Calling the function (Option 6)***

Displayed below in (Figure2.6.c.) is the program as it runs in PyCharm and in Command Prompt after the functions are called as previously mentioned.

A picture containing text, screenshot, black, several

Description automatically generated

***Figure 2.6.c.: Display Option 6 as it runs in PyCharm and Command Prompt***

## Exit The Program (Menu Option 7)

In this section we examine the programing to execute option 7 from the Menu of Options (Figure 2.7.a)

There is no processing or IO functions associated with option 7. The program simply ends when the user enters 6 and the **break** in the **While(True)** loop is called.

elif strChoice == 7:  
 print("Goodbye!")  
 EndProgram = input('\n(Press Enter to End Program)')  
 break  
 else:  
 print('\nInvalid entry. Please enter a number from 1 to 7')  
except ValueError:  
 print('\nInvalid entry. Please enter a number from 1 to 7')

***Figure 2.7.a: Calling the function (Option 7)***

Displayed below in (Figure2.7.b.) is the program as it runs in PyCharm and in Command Prompt after the loop break is called when the user enters 7.

Text

Description automatically generated

***Figure 2.7.b: Display Option 7 as it runs in PyCharm and Command Prompt***

# Summary

To recap, the seventh module introduced me to working with files and exceptions The example created for this assignment, Assignment07.py is the result of steps taken to develop a script designed to prompt the user to select from a seven-option menu and execute the program based on the selection. The program successfully allows the user to enter, display, delete, reload, and save the data in .txt and .dat files known as “pickling”. The program runs in PyCharm and Command Prompt and considers the concepts and best practices learned in this module.