Luis Valderrama

08/25/2021

IT FDN 110 A

Module 08

[GitHub Repository Module0](https://github.com/lvalderr/IntroToProg-Python-Mod08)8

Classes and Objects

# Intro

During the eighth module, I learned the concepts of working with Classes and Objects. This document covers the steps to create a script named, *Assignment08-starter.py*, designed to prompt the user to select from a four-option menu and execute the program based on the selection. This script introduces the concept of classes. The script is created using PyCharm as the Integrated Development Environment (IDE) and runs in PyCharm and Command Prompt.

# Assignment08.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 08  
# Description: Working with classes  
# Luis Valderrama, 8.17.2021,Created started script  
# Luis Valderrama, 8.21.2021,added menu options  
# ------------------------------------------------------------------------------------------- #  
  
# Objective:  
# 1. The Assignment08.py script is designed to:  
# a. Present a menu of choices for the user to select from.  
# b. Execute the program based on the choice made by the user.  
# c. Use Try/Catch Error Handling.  
# d. Use Class objects.

***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 7 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 Product functions --------------------------------------------------- #

# Fields ------------------------------------------------------------------------------------ #

# Constructor ------------------------------------------------------------------------------- #

# Properties -------------------------------------------------------------------------------- #  
# Methods ----------------------------------------------------------------------------------- #

# Step 3 - set up Class FileProcessor functions --------------------------------------------- #

# Step 4 – set up class data processor ------------------------------------------------------ #

# Presentation (Input/Output) --------------------------------------------------------------- #  
# Step 5 - set up presentation, IO functions ------------------------------------------------ #

# Main Body of Script ----------------------------------------------------------------------- #  
# Step 6 - Load data from file into a list of product objects when script starts ------------ #

# Step 7 Use of class: ---------------------------------------------------------------------- #

# Display menu of options ------------------------------------------------------------------- #

# Display current inventory ----------------------------------------------------------------- #

# Enter new data to list -------------------------------------------------------------------- #

# Save data from list to .txt --------------------------------------------------------------- #

# 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)

strChoice = "" # Captures the user option selection  
strFileName = 'products.txt' # The name of the data file  
lstOfProductObjects = [] # A list that acts as a 'table' of rows

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

## Class Products (object)

The **class Product** is set up in this assignment to process the data. The data in the class is defined as **Fields** as one of the patterns which in this case are, **strProductName** and **floatProductPrice**. The class includes other patterns such as the **Constructor (with attributes)** which is defined as **def \_\_int\_\_** with variables relating back to the fields. In this case the variables for the constructor are **self, product\_name and price\_.** The class also includes **Properties** that act as “getter” and “setters”. The “getters” are set as **@property,** while the “setters” are set as **@product\_name.setter** and **@product\_price.setter** and defined as **product\_name** and **product\_price**.Another pattern used in this class is the **Method** which defined by **def to\_string** with the **\_\_str\_\_()** method to return the class’s data as strings. (Figure 1.4)

Text

Description automatically generated

***Figure 1.4: Class Products (object)***

## Class FileProcessor (object)

This assignment uses two processing functions, save to file and read from file as outlined below in (Figure 1.5). Each function will be used for a specific process in the Menu of Options and will be called based upon the user’s input. The functions are examined in detail in section 2.0, Menu of Options.

Text

Description automatically generated

***Figure 1.5: Class FileProcessor***

## Class Data Processor

This assignment uses one data processing function that appends new data to the list (Figure 1.6). The function will be examined in more detail as we start looking at section 2.0 Menu of Options.

Text

Description automatically generated

***Figure 1.6: Class Processor***

## Class IO (Input/Outputs)

This assignment uses four additional functions as outlined below in (Figure 1.7). 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. The functions are examined in more detail as we start looking at section 2.0 Menu of Options.

Text

Description automatically generated

***Figure 1.7: Class IO (Inputs/Outputs)***

## Loading Data from Text File to the Table

When the program starts it opens the products.txt and loads the two-column content into a list held in a table named **lstOfProductObjects** as shown in (Figure 1.8.a).

The function is defined using **def** followed by the name of the function, in this case **read\_data\_from\_file** and the variable given is **file\_name** which represents the text file products.txt where the list of products is saved. The parameters are entered as well as a return which in this example is the list of items. The actual function code is entered to open the .txt file, retrieve the data, and place it in the **lstOfProductObjects** in the form of append. The .txt file is then closed.

@staticmethod  
def read\_data\_from\_file(file\_name: str, list\_of\_product\_objects: list):  
 *""" Reads data from a file into a list of object rows* ***:param*** *list\_of\_product\_objects: (list) of objects data saved to file* ***:param*** *file\_name: (string) with name of file = products.txt* ***:return****: (list) of object rows  
 """* list\_of\_product\_objects.clear()  
 try:  
 objFile = open(file\_name, "r")  
 for row in objFile:  
 list\_of\_product\_objects.append(row[0] + ', ' + row[1])  
 objFile.close()  
 except Exception as e:  
 print("There is an error")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')  
 return list\_of\_product\_objects

***Figure 1.8.a: Class FileProcessor. read\_data\_from\_file***

The **class FileProcessor. read\_data\_from\_file** functionis called before the **While** loop containing the Menu of Options. (Figure 1.8.b.). Additionally, the variable **strFileName** represents the products.txt (Figure 1.8.c) where the information is held and will be loaded to the list and displayed on the screen when the user enters option 1.

try:  
 lstOfProductObjects = FileProcessor.read\_data\_from\_file(strFileName, lstOfProductObjects)  
except FileNotFoundError:  
 IO.input\_press\_to\_continue('File not found. Please select option 3 to save data and continue.')

***Figure 1.8.b: Calling the function. read\_data\_from\_file***

Text

Description automatically generated

***Figure 1.8.c: View of products.txt content***

# Menu of Options Set Up

In this assignment, the program initiates with a menu that offers four choices as shown below in (Figure 2.0.a). The menu is defined in the **Class IO** (input/output) as **print\_menu\_of\_options.**

@staticmethod  
def print\_menu\_of\_Options():  
 *""" Display a menu of choices to the user* ***:return****: Nothing  
 """* print('''  
 Menu of Options  
 1) Display current product inventory  
 2) Add a new inventory item  
 3) Save data to file  
 4) Exit program  
 ''')  
 print()

***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 – 4 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 loop 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 4] - "))  
 print()   
 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 the break is reached. (Figure 2.0.d.).

while True:  
 try:  
 # 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 run in Command Prompt. The image below shows the Menu of Options 1 - 4 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 4] 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 Product 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 Input/Output function is defined as **print\_current\_products\_in\_list** with the variable, **list\_of\_product\_objects,** 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 with values. The **print()** is set up to “unpack” the data.

@staticmethod  
def print\_current\_products\_in\_list(list\_of\_product\_objects: list):  
 *""" Shows the current inventory* ***:param*** *list\_of\_product\_objects: (list) split data into list* ***:return****: Nothing  
 """* print()  
 print("\*\*\*\*\*\*\* The current inventory: \*\*\*\*\*\*\*")  
 for row in list\_of\_product\_objects:  
 print(row[0] + ', ' + row[1].strip())  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print()

***Figure 2.1.a.: Class IO.print\_current\_products\_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 – 4]”. 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\_products\_in\_list(lstOfProductObjects)

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.)

As mentioned in section 1.4, the function is set up with two fields, **strProductName** and **floatProductPrice**, serving as the data members of the **Class Product**. The constructors are set with initial values **def \_\_init\_\_** followed variables **product\_name** and **product\_price**, and keyword **“self”** which is used to refer data or functions in the object.

class Product(object):  
 *"""Stores data about a product:  
  
 properties:  
 product\_name: (string) with the product's name  
 product\_price: (float) with the product's standard price  
  
 changelog: (When,Who,What)  
 Luis Valderrama, 8.17.2021, Created Class  
 """* # Fields -------------------------------------------------------------------------------- #  
 strProductName = ""  
 floatProductPrice = ""  
  
 # Constructor --------------------------------------------------------------------------- #  
 def \_\_init\_\_(self, product\_name, product\_price):  
 # -- Attributes -- #  
 self.\_\_product\_name = product\_name  
 self.\_\_product\_price = product\_price

***Figure 2.2a.: Class Product***

The data processing function is defined as **add\_data\_to\_list** with the variables, **list\_of\_product\_objects, product\_name** and **product\_price** in the form of strings. The function itself takes the product and price entered by the user and appends the **list\_of\_product\_objects** that holds the data. The function **returns** a value in the form of an expression immediately after the user enters the new data.

class DataProcessor:  
 *""" Performs other processing data  
  
 changelog: (When,Who,What)  
 Luis Valderrama, 8.17.2021, Created Class  
 """* @staticmethod  
 def add\_data\_to\_list(list\_of\_product\_objects: list, product\_name: str, product\_price: str):  
 newProduct = product\_name, product\_price  
 list\_of\_product\_objects.append(newProduct)  
 return list\_of\_product\_objects

***Figure 2.2.b.: Class Data Processor***

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 – 4]”. The next function called gets data entered by the user. (Figure 2.2.c.)

elif strChoice == 2:  
 lstData = IO.input\_product\_data()  
 DataProcessor.add\_data\_to\_list(lstOfProductObjects, lstData[0], lstData[1])  
 IO.print\_current\_products\_in\_list(lstOfProductObjects)  
 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***

## Saving Data to File (Menu Option 3)

In this section we examine the programing to execute option 4 from the Menu of Options (Figure 2.3.a)

The processing function is defined as **save\_data\_to\_file** with the variable, **file\_name** and **list\_of\_product\_objects** in the form of string. The function itself opens the Products.txt and saves the data from the products and prices list. The program then closes the .txt file and prints the message with confirmation the data has been saved in the Products.txt.

@staticmethod  
def save\_data\_to\_file(file\_name: str, list\_of\_product\_objects: list):  
 *""" Write data to a file from a list of object rows* ***:param*** *file\_name: (string) with name of file = products.txt* ***:param*** *list\_of\_product\_objects: (list) of objects data saved to file* ***:return****: (bool) with status of success status  
 """* status = False  
 try:  
 objFile = open(file\_name, 'w')  
 for row in list\_of\_product\_objects:  
 objFile.write(str(row[0].strip() + ', ' + row[1].strip()) + '\n') # calls \_\_str\_\_()  
 objFile.close()  
 status = True  
 except Exception as e:  
 print("There is an error")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')  
 return status

***Figure 2.3.a: Class FileProcessor.save\_data\_to\_file (Option 3)***

The function previously described is called within the **while(True)** loop when the user enters “4” in the **FileProcessor.save\_data\_to\_file**. The program also calls the **IO.print\_current\_inventory\_in\_list** function which displays the current products and values. (Figure 2.3.b.)

elif strChoice == 3:  
 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':  
 FileProcessor.save\_data\_to\_file(strFileName, lstOfProductObjects)  
 print('\nYour data is saved to', strFileName)  
 IO.input\_press\_to\_continue()  
 continue

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

Displayed below in (Figure2.3.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 2.3.c: View of the Option 3 as it runs in PyCharm and Command Prompt***

To view the saved data simply open the products.txt. (Figure 2.3.d)

Text

Description automatically generated

***Figure 2.3.d: View of the products.txt content***

## Exit The Program (Menu Option 4)

In this section we examine the programing to execute option 4 from the Menu of Options (Figure 2.4.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 == 4:  
 print("Goodbye!")  
 EndProgram = input('\n(Press Enter to End Program)')  
 break  
 else:  
 print('\nInvalid entry. Please enter a number from 1 to 4')  
except ValueError:  
 print('\nInvalid entry. Please enter a number from 1 to 4')

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

Displayed below in (Figure2.4.b.) is the program as it runs in PyCharm and in Command Prompt after the loop break is called when the user enters 4.

Text

Description automatically generated

***Figure 2.4.b: Display Option 4 as it runs in PyCharm and Command Prompt***

# Summary

To recap, the eighth module introduced me to working with classes and objects. The example created for this assignment, Assignment08.py is the result of steps taken to develop a script designed to prompt the user to select from a four-option menu and execute the program based on the selection. The program successfully allows the user to enter, display, read, and save the data in a .txt file. The program runs in PyCharm and Command Prompt and considers the concepts and best practices learned in this module.