Luis Valderrama

08/08/2021

IT FDN 110 A

Module 06

Link to GitHub Repository: <https://github.com/lvalderr/IntroToProg-Python-Mod06>

Link to blog: <https://lvalderr.github.io/IntroToProg-Python-Mod06/>

Functions

# Intro

During the sixth module, I learned the concepts of working with Functions. This document covers the steps to create a script named, *Assignment06\_Starter.py* (Python script), designed to prompt the user to select from a six-option menu and execute the program based on the selection. The six options are 1) Display current tasks, 2) Add a new task, 3) Remove an existing task, 4) Save data to file, 5) Reload data from file, and 6) Exit program. The script is designed to run in PyCharm and Command Prompt, applying concepts learned throughout this module. The following steps were taken to create the Python Script using PyCharm as the Integrated Development Environment (IDE).

# 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 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"  
# ChangeLog (Who,When,What):  
# LValderrama,7.31.2021,Created started script

# LValderrama, 8.07.2021, Modified writing to file, print from list option 4 and 1

# LValderrama, 8.07.2021, Modified remove from list option 3

# ------------------------------------------------------------------------------------------- #  
  
# Objective:  
# The Assignment06.py script is designed to:  
# 1. Present a menu of choices for the user to select from.  
# 1) Display current data  
# 2) Add a new Task  
# 3) Remove an existing Task  
# 4) Save Data to File  
# 5) Reload Data from File  
# 6) Exit Program

# 2. Execute the program based on the choice made by the user

***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 4 core steps that may expand into sub-steps as the script develops.

# Data --------------------------------------------------------------------------------------------------------------- #  
# Declare variables and constants

# Processing Data ---------------------------------------------------------------------------------------------------- #

# Presentation, IO: Input and Output --------------------------------------------------------------------------------- #

# Main Body of the Script -------------------------------------------------------------------------------------------- #  
# Step 1 - When the program starts, load data from ToDoFile.txt

# Step 2 - Display a menu of choices to the user

# Step 3 Display menu of options

# Step 4 Process user's menu choice

***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)

# Declare variables and constants  
strFileName = "ToDoFile.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 {Task,Priority}  
lstTable = [] # A list that acts as a 'table' of rows  
strChoice = "" # Captures the user option selection  
strTask = "" # Captures the user task data  
strPriority = "" # Captures the user priority data  
strStatus = "" # Captures the status of an processing functions

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

## Class Processor

This assignment uses four 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. These functions are intended to process the data based on the user’s input. Then, I insert the variables, parameters for each function.

Note: 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 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.

Note: 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 ToDoList.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\_file** and the variable given is **file\_name** which represents the text file ToDoList.txt where the list of tasks 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**.

@staticmethod  
def read\_data\_from\_file(file\_name):  
 *""" Reads data from a file into a list of dictionary rows* ***:param*** *file\_name: (string) strFileName = ToDoFile.txt* ***:return****: (list) dictionary rows  
 """* lstTable = []  
 objFile = open(file\_name, "r")  
 for row in objFile:  
 if "," in row:  
 lstRow = row.split(",")  
 dicRow = {"Task": lstRow[0].strip(), "Priority": lstRow[1].strip()}  
 lstTable.append(dicRow)  
 objFile.close()  
 return lstTable

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

The **Class Processor. read\_data\_from\_file** functionis called outside of the While loop that will contain the Menu of Options. (Figure 1.6.b.). Additionally, the variable **strFileName** represents the ToDoFile.txt where the information is held (Figure 1.6.c).

# Step 1 - When the program starts, load data from ToDoFile.txt  
lstTable = Processor.read\_data\_from\_file(strFileName)

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

A screenshot of a computer

Description automatically generated

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

# Menu of Options Set Up

In this assignment, the program initiates with a menu that offers six choices. 1) Show current tasks, 2) Add a new task, 3) Remove an existing task, 4) Save data to file, 5) Reload data from file, and 6) Exit program. The menu is defined in the **Class IO** (input/output) as **print\_menu\_tasks**, without variables. (Figure 2.0.a)

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

***Figure 2.0.a: Class IO. print\_menu\_tasks***

An **input()** function is defined as **IO**.**input\_menu\_choice()** in the form of a string. This input is established to get a choice from 1 – 6 from the user (Figure 2.0.b). The choice will then activate the corresponding function and the program will process the data accordingly.

@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 6] - ")).strip()  
 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.')

return choice

***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):  
# Step 3 Display menu of options

IO.print\_menu\_Tasks()  
 strChoice = IO.input\_menu\_choice()

***Figure 2.0.d: Calling the function***

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 - 6 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 6] 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 Tasks (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\_tasks\_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 tasks and priorities. The print() is set up to “unpack” the data.

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

***Figure 2.1.a.: Class IO.print\_current\_tasks\_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 – 6]”. 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.)

# Step 4 Process user's menu choice  
 if strChoice.strip() == '1':  
 IO.print\_current\_tasks\_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 Task (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, task, priority** in the form of strings. The function itself takes the task and priority 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 tasks and priorities.

@staticmethod  
def add\_data\_to\_list(list\_of\_rows, task, priority):  
 *""" Adds task and priorities to list* ***:type*** *list\_of\_rows:* ***:param*** *list\_of\_rows: (list) you want filled with file data* ***:param*** *task: (string) to do item* ***:param*** *priority: (string) the priority of the task* ***:return****: (list) of task and priorities  
 """* dicRow = {"Task": str(task), "Priority": str(priority)}  
 list\_of\_rows.append(dicRow)  
 return list\_of\_rows, 'Success'

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

The Input/Output function is defined as **input\_new\_task\_and\_priority** without parameters. The function gets two inputs, Task and Priority from the user, and it **returns** the entries made by the user. (Figure 2.2.b)

@staticmethod  
def input\_new\_task\_and\_priority():  
 *""" Adds new task and priority entered by the user* ***:return****: (string) Task and its corresponding priority  
 """* task = str(input("Enter a New Task: ")).strip()  
 priority = str(input("Enter Priority (Top, Med, Low): ")).strip()  
 print()  
 return task, priority

***Figure 2.2.b: IO.input\_new\_task\_and\_priority (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 – 6]”. The next function called is **IO.input\_new\_task\_and\_priority()** 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 task and its priority. The program calls **IO.print\_current\_tasks\_in\_list** function which now displays the list updated with the new task and priority 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\_task\_and\_priority()  
 Processor.add\_data\_to\_list(lstTable, tplData[0], tplData[1])  
 IO.print\_current\_tasks\_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 Task (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 tasks and priorities, **lstTable**. If the input is matched with an item in the list, then the function removes the task and priority 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 tasks from list* ***:param*** *list\_of\_rows: (list) you want filled with file data* ***:return****" (List) of tasks removed  
 """* while len(lstTable) > 0:  
 print("Current Task list:\n", "\n".join([f"\t{item['Task']}" for item in lstTable])) # Display current data  
 term = input("Enter the Task you want to delete or type 'exit' to return to Menu of Options: ")  
 if term.lower() == "exit":  
 break  
 for task in lstTable:  
 if term in task["Task"]: # If the task entered by the user matches the name in the list then proceed  
 print(f"Removing {task['Task']}...") # Print a message "item found" and processing removal  
 list\_of\_rows.remove(task) # The item is removed from the table  
 break  
 else:  
 print(f"Task not found in list: {term}") # Displays message indicating the task 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 tasks and priorities entered by the user  
  
 :term: (string) this is the item the user wants to remove* ***:return****: (string) Task and its corresponding priority  
 """* term: input("Confirm the Task you want to delete or type 'exit' to return to Menu of Options: ")

***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** with the variable, **file\_name** and **list\_of\_rows** in the form of string. The function itself opens the ToDoFile.txt and saves the data from the tasks and priorities list. The program then closes the .txt file and prints the message with confirmation the data has been saved in the ToDoFile.txt.

@staticmethod  
def write\_data\_from\_list(file\_name, list\_of\_rows):  
 *"""* ***:param*** *file\_name: (string) strFileName = ToDoFile.txt* ***:param*** *list\_of\_rows: (list) you want filled with file data  
 """* objFile = open(file\_name, 'w')  
 for row in list\_of\_rows:  
 objFile.write(row["Task"] + "," + row["Priority"] + "\n")  
 objFile.close()  
 print('\nYour data is saved to', file\_name)

***Figure 2.4.a: Class Processor.write\_data\_from\_list (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**. The program also calls the **IO.print\_current\_tasks\_in\_list** function which displays the current tasks and priorities. The last function called is **IO.input\_press\_to\_continue()** in order to pause the program until the user presses Enter to proceed. (Figure 2.4.b.)

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(strFileName, lstTable)  
 print('\nYour data is saved to', strFileName)  
 IO.print\_current\_tasks\_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 ToDoFile.txt. (Figure 2.4.d)

A screenshot of a computer

Description automatically generated

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

## Reload Data from 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 ToDoFile.txt into the list table. The program then displays the data by way of calling the function **IO.print\_current\_tasks\_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 tasks!")  
 strAcceptOrDecline = input("Update data without saving? type 'y' or 'n' ")  
 if strAcceptOrDecline.lower() == 'y':  
 lstTable.clear()  
 lstTable = Processor.read\_data\_from\_file(strFileName)  
 IO.print\_current\_tasks\_in\_list(lstTable)  
 else:  
 input("File no updated. Press Enter to go back to Menu of Options")  
 IO.print\_menu\_Tasks()  
 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***

## Exit The Program (Menu Option 6)

In this section we examine the programing to execute option 6 from the Menu of Options (Figure 2.6.a)

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

elif strChoice == '6':  
 print("Goodbye!")  
 EndProgram = input('\n(Press Enter to End Program)')  
 break

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

Displayed below in (Figure2.6.b.) is the program as it runs in PyCharm and in Command Prompt after the loop break is called when the user enters 6.

Text

Description automatically generated

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

# Summary

To recap, the sixth module introduced me to working with Functions. The example created for this assignment, Assignment06\_Starter.py is the result of steps taken to develop a script designed to prompt the user to select from a six-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 a .txt file. The program runs in PyCharm and Command Prompt and considers the concepts and best practices learned in this module.