Name: Gail Provancha

Date: August 15, 2023

Course: IT FDN 110 A Foundations of Programming: Python

Assignment: Assignment06

GitHub: https://github.com/provgl1/IntroToProg-Python-Mod06

Working with Functions and Classes

Introduction

This document will go over the steps needed to write a Python Script that provides a menu for the user to make selections that can add, delete, and save to a "to do list". This script will build upon Assignment05 by using functions and classes. A pre-existing starter script called "Assignment06_Starter.py"will be used.

Sections of the Script

The script will include a header, which will include a title, description and a change log. The header is to provide information regarding the script for individuals regarding the purpose of the script and when and who made modifications.

The next section will be used to declare global variables, which are variables that can be used both inside and outside of a function and can be called from multiple sections of the script.

Then there will be a section for our functions which are grouped under Processing and Presentation. This makes the code reusable and easier to update.

And lastly there is the Main Body of the script which calls the different functions and executes that block of code under those functions (Figure 1).

```
1    # ------------------------------------------------------------------ #
2    # Title: Assignment 06
3    # Description: Working with functions in a class,
4    #              When the program starts, load each "row" of data
5    #              in "ToDoToDoList.txt" into a python Dictionary.
6    #              Add to each dictionary "row" to a python list "table"
7    # ChangeLog (Who,When,What):
8    # RRoot,1.1.2030,Created started script
9    # Gail Provancha,8.14.2023,Modified code to complete assignment 06
10   # ------------------------------------------------------------------ #
11
12   # Data ------------------------------------------------------------- #
13   # Declare variables and constants
14   file_name_str = "ToDoFile.txt"  # The name of the data file
15   file_obj = None  # An object that represents a file
16   row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priority}
17   table_lst = []  # A list that acts as a 'table' of rows
18   choice_str = ""  # Captures the user option selection
19
20   # Processing  ------------------------------------------------------ #
21
22   # Presentation (Input/Output)  ------------------------------------ #
23
24   # Main Body of Script  -------------------------------------------- #
25
26       # Step 1 - When the program starts, Load data from ToDoFile.txt.
27       # Step 2 - Display a menu of choices to the user
```

Figure 1: Sections of Script

Classes and Functions

The Functions are grouped into two sections, Processing and Presentation. The grouping is called a "class", which will keep the similar type functions in one location. The two classes that were created are "Processing" and "I/O". The Main script will call the functions that are under the class, passing values to the arguments /parameters for execution.

Class Processor

There are 4 functions under the "Processor" class to include: 1) read data from a file, 2) add data to a list, 3) remove data from a list, and 4) write data to a file.

Step 1 – Read data from a file

The function called "def read_data_from_file" will receive two parameters from the main script, "file_name" and "list_of_rows" perform the statements under the function, and then return "list_of_rows" which is the result of the execution of the function. This involves opening the file, looping thru the rows, splitting each row based on a "," saving the information to a dictionary row, and then appending the dictionary row to a list (Figure 2).

```python
21      # Processing  ------------------------------------------------------ #
        4 usages
22      class Processor:
23          """  Performs Processing tasks """
24

            1 usage
25          @staticmethod
26          def read_data_from_file(file_name, list_of_rows):
27              """ Reads data from a file into a list of dictionary rows
28
29              :param file_name: (string) with name of file:
30              :param list_of_rows: (list) you want filled with file data:
31              :return: (list) of dictionary rows
32              """
33              list_of_rows.clear()   # clear current data
34              file = open(file_name, "r")
35              for line in file:
36                  task, priority = line.split(",")
37                  row = {"Task": task.strip(), "Priority": priority.strip()}
38                  list_of_rows.append(row)
39              file.close()
40              return list_of_rows
```

Figure 2: Reading File into List

Step 2 – Add data to a list

The function called "def add_data_to_list" will receive three parameters from the main script, "task", "priority", and "list_of_rows" perform the statements under the function, and then return "list_of_rows" which is the result of the execution of the function.

The "task" and "priority" inputs are added to a dictionary row and then appended the "list_of_rows" table using the append() function (Figure 3).

```
1    @staticmethod
2    def add_data_to_list(task, priority, list_of_rows):
3        """ Adds data to a list of dictionary rows
4
5        :param task: (string) with name of task:
6        :param priority: (string) with name of priority:
7        :param list_of_rows: (list) you want to add more data to:
8        :return: (list) of dictionary rows
9        """
10       row = {"Task": str(task).strip().title(), "Priority": str(priority).strip().title()}
11       list_of_rows.append(row)  # append dictionary row to list
12       return list_of_rows
13
```

Figure 3: appending to the list

Step 3 – Remove data from a list

The function called "remove_data_from_list" will receive two parameters from the main script, "task" and "list_of_rows" perform the statements under the function, and then return "list_of_rows" which is the result of the execution of the function. This process will loop through each row from the list, checking to see if matches the passed parameter "task" and remove the row if it does (Figure 4).

```
55   @staticmethod
56   def remove_data_from_list(task, list_of_rows):
57       """ Removes data from a list of dictionary rows
58
59       :param task: (string) with name of task:
60       :param list_of_rows: (list) you want filled with file data:
61       :return: (list) of dictionary rows
62       """
63
64       for row in list_of_rows:
65           t, p = dict(row).values()  # unpack the dictionary row in the list of rows
66           if t.lower() == task.lower():
67               list_of_rows.remove(row)
68       return list_of_rows
```

Figure 4: remove data from list

Step 4 – Write data to a file

The function called "write_data_to_file" will receive two parameters from the main script, "file_name" and "list_of_rows" perform the statements under the function, and then return "list_of_rows" which is the result of the execution of the function.

This will involve opening the file, looping through the rows in the table, writing each row to the file, and closing the file, which saves the information (Figure 5).

```
1
2    @staticmethod
3    def write_data_to_file(file_name, list_of_rows):
4        """ Writes data from a list of dictionary rows to a File
5
6        :param file_name: (string) with name of file:
7        :param list_of_rows: (list) you want filled with file data:
8        :return: (list) of dictionary rows
9        """
10
11       file = open(file_name, 'w')  # create text file
12       for row in list_of_rows:  # loop through the rows in the table
13           file.write(row["Task"].title() + "," + row["Priority"].title() + "\n")
14       file.close()  # close the text file
15       return list_of_rows
16
17
```

Figure 5: write to file

Class I/O (input and output)

There are 5 functions under the "I/O" class which corresponds to the menu selections: 1) output menu tasks, 2) input menu choice, 3) output current tasks in list, 4) input new task and priority, and 5) input task to remove.

Step 1 – Output Menu Task

The function displays the menu to the user (Figure 5).

```
1  class IO:
2      """ Performs Input and Output tasks """
3
4      @staticmethod
5      def output_menu_tasks():
6          """  Display a menu of choices to the user
7
8          :return: nothing
9          """
10         print('''
11             Menu of Options
12             1) Add a new Task
13             2) Remove an existing Task
14             3) Save Data to File
15             4) Exit Program
16             ''')
17         print()  # Add an extra line for looks
```

Figure 6: Menu

Step 2 – Input Menu Choice

The function saves the selection made by the user to a variable called "choice" (Figure 7).

```
        @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 4] - ")).strip()
            print()  # Add an extra line for looks
            return choice
```

Figure 7: Choice

Step 3 – Output Current tasks in list

The function displays the current tasks in the list (Figure 8).

```
        @staticmethod
        def output_current_tasks_in_list(list_of_rows):
            """ Shows the current Tasks in the list of dictionaries rows

            :param list_of_rows: (list) of rows you want to display
            :return: nothing
            """
            print("******* The current tasks ToDo are: *******")
            for row in list_of_rows:
                print(row["Task"] + " (" + row["Priority"] + ")")
            print("****************************************")
            print()  # Add an extra line for looks
```

Figure 8: current tasks

Step 4 – Input new task and priority

The function requests the user to name a task and its priority then saves those inputs as variables "task" and "priority" (Figure 9).

```
        @staticmethod
        def input_new_task_and_priority():
            """  Gets task and priority values to be added to the list

            :return: (string, string) with task and priority
            """

            task = str(input("Please add a task: ")).strip()  # strip to remove any added spaces
            priority = str(input("What is the priority of that task?: ")).strip()
            return task, priority
```

Figure 9: current tasks

Step 4 – Input task to remove

The function requests the user to name a task to be removed from the list. The input is saved in the variable "task" (Figure 10).

```
        @staticmethod
        def input_task_to_remove():
            """  Gets the task name to be removed from the list

            :return: (string) with task
            """

            task = str(input("What task would you like to remove? : ")).strip()
            return task
```

Figure 10: remove task

Main Body of Script

The main body of the script will call the different functions that were created earlier, based on the menu choices the user has made (Figure 11).

```
161
162    # Main Body of Script  ------------------------------------------------- #
163
164
165    # Step 1 - When the program starts, Load data from ToDoFile.txt.
166    Processor.read_data_from_file( file_name=file_name_str, list_of_rows=table_lst)  # read file data
167
168    # Step 2 - Display a menu of choices to the user
169    while (True):
170        # Step 3 Show current data
171        IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the list/table
172        IO.output_menu_tasks()  # Shows menu
173        choice_str = IO.input_menu_choice()  # Get menu option
174
175        # Step 4 - Process user's menu choice
176        if choice_str.strip() == '1':  # Add a new Task
177            task, priority = IO.input_new_task_and_priority()
178            table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
179            continue  # to show the menu
180
181        elif choice_str == '2':  # Remove an existing Task
182            task = IO.input_task_to_remove()
183            table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
184            continue  # to show the menu
185
186        elif choice_str == '3':  # Save Data to File
187            table_lst = Processor .write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
188            print("Data Saved!")
189            continue  # to show the menu
190
191        elif choice_str == '4':  # Exit Program
192            print("Goodbye!")
193            break  # by exiting loop
194
```

Figure 11: Main Script

Testing the Script

We will be testing the script in both PyCharm and Command Prompt.

Pycharm

First, let's test the script by running it in PyCharm (Figure 12).

Figure 12a: Add a new task

Figure 12b: Remove an existing task

Figure 12c: Save the File

Figure 12d: Exit

And then open up "ToDoFile.txt" file to verify the results are what we would expect (Figure 13).

Figure 13: text file

CMD line

Now the script will be run from Command Prompt to make sure it also runs as expected. Open a command prompt window "CMD". Type in "cd" and then the pathway to the directory where the python script is saved "C:_PythonClass\Module06\Assignment06". Click Enter. Next type "python" followed by the name of the python script "Assignment06.py". Click "Enter" again to see the script run. Provide the necessary inputs as if you are the user and verify the result is what you expected (Figure 14).

```
C:\Users\prova>cd "C:\_PythonClass\Module06\Assignment06

C:\_PythonClass\Module06\Assignment06>python "Assignment06.py"
******* The current tasks ToDo are: *******
Mow Lawn (Low)
Washing Cat (Medium)
Washing Windows (Low)
Bath For Dog (Medium)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program

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

Please add a task: Clean Stove
What is the priority of that task?: low
******* The current tasks ToDo are: *******
Mow Lawn (Low)
Washing Cat (Medium)
Washing Windows (Low)
Bath For Dog (Medium)
Clean Stove (Low)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program

Which option would you like to perform? [1 to 4] - 2

What task would you like to remove? : Washing cat
******* The current tasks ToDo are: *******
Mow Lawn (Low)
Washing Windows (Low)
Bath For Dog (Medium)
Clean Stove (Low)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program

Which option would you like to perform? [1 to 4] - 3

Data Saved!
******* The current tasks ToDo are: *******
Mow Lawn (Low)
Washing Windows (Low)
Bath For Dog (Medium)
Clean Stove (Low)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Exit Program

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

Goodbye!
```
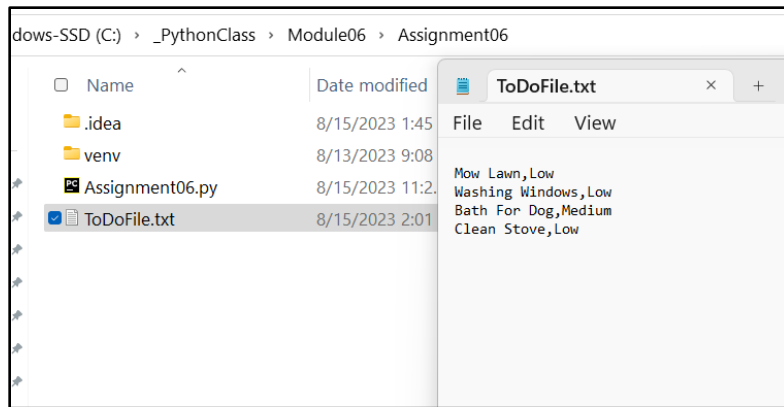
Figure 14: running script in Command Line

Open up the "ToDoFile.txt" file to verify the results have changed based on the new inputs (Figure 15).

Figure 15: Testing in Command Line, results

Summary

In summary, this document goes over the steps needed to write a Python script that provides a menu where a user can make different selections to update a to-do-list, including adding tasks, removing tasks and saving the latest version of their list. The script introduced the use of functions and classes to allow for the script to be organized in logical sections which allows for easier reading and reusability of some code.