Michelle Diaz

February 21, 2023

IT FDN 110 A Wi 23: Foundations of Programming: Python

Assignment06

https://github.com/mdiaz1122/IntroToProg-Python-Mod06

Assignment 06: Functions

Introduction

This document outlines the process used to create a script that provides the user several options for managing a To-Do list (Figure 1). This assignment is nearly identical to Assignment05 except this time we organize the program better using re-usable blocks of code called Functions.

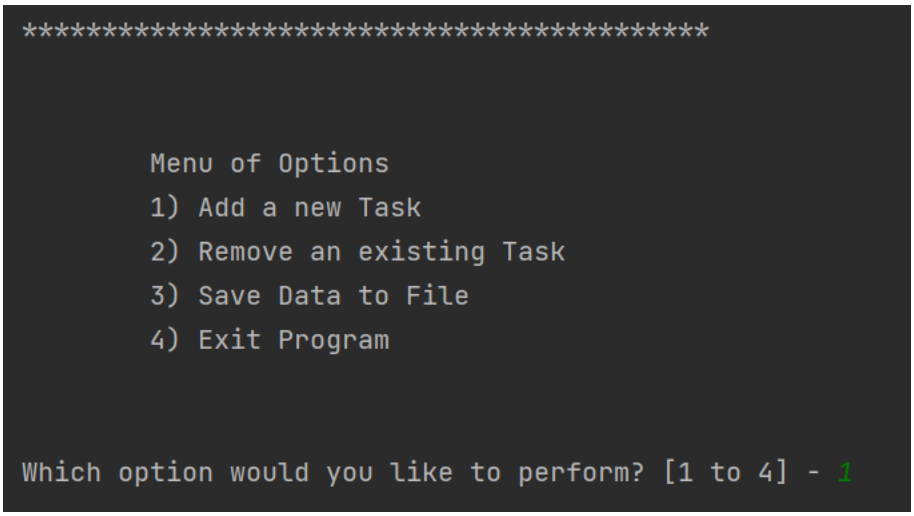

Figure 1: List of options user has to manage the Task list

The menu is provided after every operation except when the user chooses to exit the program. When the program is started, there is a check done for an existing file and any pre-existing data is populated to the list variable and available for appending, removal, and reading of Tasks previously saved.

Creating the Program

Since the code used for this program is very similar to the Assignment05 code, I am going to focus more on the differences between Assignment05 and Assignment06 as well as any blocks of code I added to the 'Assignment06_Starter.py' file to execute this assignment.

Organizing the Program

For this assignment, separation of concerns is utilized as well as error handling and functions. This made it much easier to manage the code in the program since the functions help isolate the specific executables

and the separation of concerns allowed for more efficient handling of the data, processing, and input/output concerns. Additionally, the error handling is particularly useful here since the text file does not exist if the program has never been run before. Therefore, upon program initialization the text file is queried and if it does not exist then a 'FileNotFoundError' occurs. If this error occurs, instead of the program stopping and giving an error the try/except statement is passed and program continues as usual (Figure 2).

```
@staticmethod
def read_data_from_file(file_name, list_of_rows):
    """ Reads data from a file into a list of dictionary rows

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    list_of_rows.clear()  # clear current data
    try:
        file = open(file_name, "r")
        for line in file:
            task, priority = line.split(",")
            row = {"Task": task.strip(), "Priority": priority.strip()}
            list_of_rows.append(row)
        file.close()
    except FileNotFoundError:  # Adds provision for when no file exists yet
        pass
    return list_of_rows
```

Figure 2: Reading existing data into the list_of_rows variable as dictionary objects

Data

The data section lists any variables and constants being used in the program (Figure 3). Here we acknowledge that these are global variables and constants since they exist outside of any one class or function. If the same variable name is used inside of a function to set a new value, it will not change the global variable and instead will shadow the global variable.

```
# Data -------------------------------------------------------------------- #
# Declare variables and constants
file_name_str = "ToDoFile.txt"  # The name of the data file
file_obj = None  # An object that represents a file
row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priority}
table_lst = []  # A list that acts as a 'table' of rows
choice_str = ""  # Captures the user option selection
```

Figure 3: Global variables declaration

Processing

Function: read_data_from_file

Although there wasn't a "TO DO" flag at this function in the starter code, I modified the program to include a Try/Except provision that checks for the file instead of erring out if the file does not exist (Figure 4).

```
@staticmethod
def read_data_from_file(file_name, list_of_rows):
    """ Reads data from a file into a list of dictionary rows

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    list_of_rows.clear()  # clear current data
    try:
        file = open(file_name, "r")
        for line in file:
            task, priority = line.split(",")
            row = {"Task": task.strip(), "Priority": priority.strip()}
            list_of_rows.append(row)
        file.close()
    except FileNotFoundError:  # Adds provision for when no file exists yet
        pass
    return list_of_rows
```

Figure 4: read_data_from_file with Try/Except added

Function: add_data_to_list

This function was easy to complete since I had already done this in the previous assignment and just needed to fill in the append operation in the starter code (Figure 5). I did also get notified in PyCharm that the variables 'task' and 'priority' were shadowing global variables. For this program it does not really matter since I am not looking to use them outside of the local function or change the global variable definition for the operations I am performing.

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    """ Adds data to a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority:
    :param list_of_rows: (list) you want to add more data to:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
    # TODO: Add Code Here!
    list_of_rows.append(row)

    return list_of_rows
```

Figure 5: add_data_to_list with append operation

Function: remove_data_from_list

Completing this function was straightforward since I did the same thing in the previous assignment but now the operation is packaged in a function (Figure 6).

```
@staticmethod
def remove_data_from_list(task, list_of_rows):
    """ Removes data from a list of dictionary rows

    :param task: (string) with name of task:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    # TODO: Add Code Here!
    for line in range(len(list_of_rows)):
        if list_of_rows[line]['Task'] == str(task):
            del list_of_rows[line]
    return list_of_rows
```

Figure 6: remove_data_from_list with search and delete code added

Function: write_data_to_file

Completing this function was also straightforward since I did the same thing in the previous assignment but now the operation is packaged in a function (Figure 7).

```
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data from a list of dictionary rows to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    # TODO: Add Code Here!
    text_file = open(file_name, "w")
    for count in list_of_rows:
        text_file.write(str(count["Task"]) + ',' + str(count["Priority"]) + '\n')
    text_file.close()
    return list_of_rows
```

Figure 7: write_data_to_file with line by line writing code added

Presentation

Function: input_new_task_and_priority

For this function, I could have defined both variables in one line of code but for simplicity of viewing I broke it into two lines. This is just like the previous assignment task but this time, it is packaged in a

function and two variables are returned as opposed to just one like all of the other functions in the program (Figure 8).

```
@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
    """
    pass  # TODO: Add Code Here!
    task_in = str(input("Task Name: ")).strip()
    priority_in = str(input("Priority: ")).strip()
    return task_in, priority_in
```

Figure 8: input_new_task_and_priority with user prompt and variable definition code added

Function: input_task_to_remove

This function was simple to complete since it is just asking the user for the task name of the entry they want deleted from the To-Do list (Figure 9).

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

    :return: (string) with task
    """
    pass  # TODO: Add Code Here!
    task_del = str(input("Enter task to be deleted: ")).strip()
    return task_del
```

Figure 9: input_task_to_remove with user prompt code added

Main Body

Although no modifications were required in the main body of the program, it helped my understanding of what output was expected from each function by seeing how the functions were being used. For example, if not just one value was expected to be returned then two variables would be defined by the calling of a function. This is demonstrated when the add_data_to_list function is called and ‘task’ and ‘priority’ are being defined. The add_data_to_list function returns two values with the line of code ‘return task_in, priority_in’ (Figure 10).

```
# Step 4 - Process user's menu choice
if choice_str.strip() == '1':  # Add a new Task
    task, priority = IO.input_new_task_and_priority()
    table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
    continue  # to show the menu
```

Figure 10: Main body of code for adding a new task

Testing the Program

As with the previous assignment, I made sure each block of code worked prior to moving on to the next function. Once I made sure everything was working as expected, I ran the whole program in PyCharm and the Windows Command Prompt and verified the overall functionality of the program (Figures 11 thru 23). In general, there weren't many errors aside from typos since the execution of tasks was already figured out in the previous assignment.

```
C:\Users\mdiaz\venv\Scripts\python.exe C:\_PythonClass\Assignment06\Assignment06_Starter.py
******* The current tasks ToDo are: *******
*****************************************


        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

Task Name: Shower
Priority: 8
******* The current tasks ToDo are: *******
Shower (8)
*****************************************
```

Figure 11: Program execution in PyCharm

```
        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

Task Name: Laundry
Priority: 4
******* The current tasks ToDo are: *******
Shower (8)
Laundry (4)
*****************************************
```

Figure 12: Program execution in PyCharm

```
        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

Task Name: Lawn
Priority: 6
******* The current tasks ToDo are: *******
Shower (8)
Laundry (4)
Lawn (6)
*****************************************
```

Figure 13: Program execution in PyCharm

```
        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

Enter task to be deleted: Lawn
{'Task': 'Shower', 'Priority': '8'}
{'Task': 'Laundry', 'Priority': '4'}
{'Task': 'Lawn', 'Priority': '6'}
******* The current tasks ToDo are: *******
Shower (8)
Laundry (4)
*****************************************
```

Figure 14: Program execution in PyCharm

```
        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

Enter task to be deleted: Laundry
{'Task': 'Shower', 'Priority': '8'}
{'Task': 'Laundry', 'Priority': '4'}
******* The current tasks ToDo are: *******
Shower (8)
*****************************************
```

Figure 15: Program execution in PyCharm

```
        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

Task Name: Walk Dog
Priority: 10
******* The current tasks ToDo are: *******
Shower (8)
Walk Dog (10)
*****************************************
```

Figure 16: Program execution in PyCharm

```
        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: *******
Shower (8)
Walk Dog (10)
*****************************************
```

Figure 17: Program execution in PyCharm

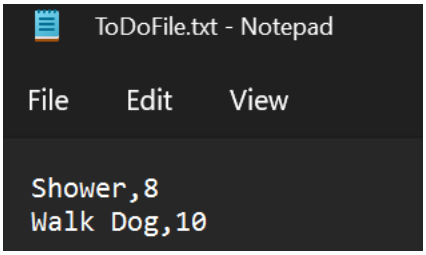

Figure 18: Program execution in PyCharm resulting text file

```
        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!

Process finished with exit code 0
```

Figure 19: Program execution in PyCharm

```
C:\_PythonClass\Assignment06>Functions.py
******* The current tasks ToDo are: *******
Shower (8)
Walk Dog (10)
*****************************************


        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

Task Name: Laundry
Priority: 6
******* The current tasks ToDo are: *******
Shower (8)
Walk Dog (10)
Laundry (6)
*****************************************
```

Figure 20: Program execution in Windows Command Prompt

```


        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: *******
Shower (8)
Walk Dog (10)
Laundry (6)
*****************************************
```

Figure 21: Program execution in Windows Command Prompt

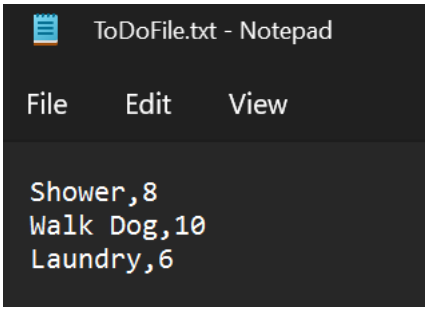

Figure 22: Program execution in Windows Command Prompt resulting text file

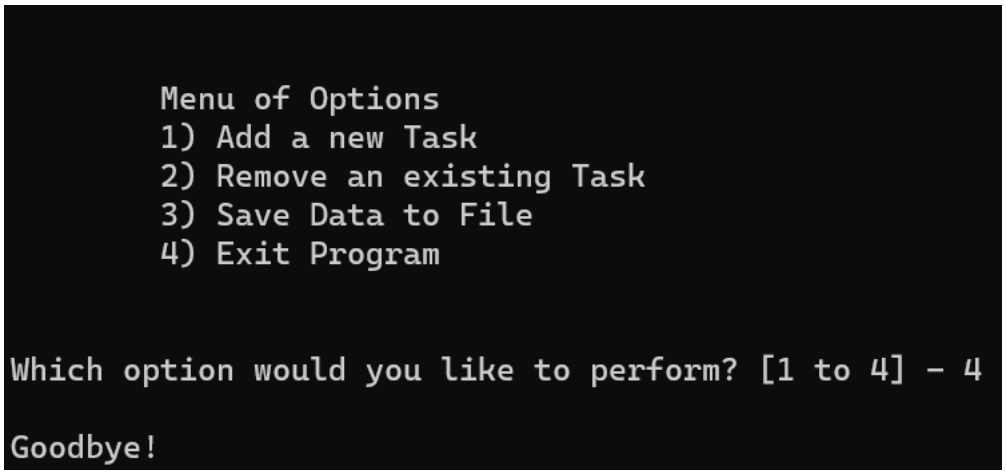

Figure 23: Program execution in Windows Command Prompt

Summary

In summary, this assignment was straightforward for me since I have prior experience with functions and passing variables to and from them and most of the programming tasks required were already programmatically figured out in the last assignment. I am glad to see that we are moving into using classes and functions now since this is a much more organized way to write a program. It's nice to be able to use the separation of concerns which is a new concept for me because this just further helps organize the program. Classes and functions are great because they make code easier to follow and modify at a later date.