

Omar Lozano

May 24, 2022

Foundations of Programming: Python

Assignment 06

<https://github.com/omega609/IntroToProg-Python-Mod6>

Working with Functions

Introduction

For this assignment I used functions to create a program to store dictionaries of tasks and priorities. Some of the code was provided by our instructor and I was to provide the missing code that would ultimately allow a user to add, remove, view, or save data to a text file. As most of the code was provided, I will only be explaining the areas where my custom code was added.

Creating the Script

This script started with pseudo code and an outline of the functions needed to complete this assignment. I began by updating the changelog and with the declaration of variables (figure 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):
# RRoot,1.1.2030,Created started script
# Omar Lozano,05/24/2022,Modified code to complete assignment 06
# ----- #
# 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 1: Update name and declare global variables

The first part of the script requiring code was the “add_data_to_list” function. This function takes in three parameters: a task, a priority and a list (list_of_rows) and adds new data to the list of dictionary elements that are loaded at the start of this program. The new list item is added using the append() method, and then the new list is returned (figure 2).

```

@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 filled with file data:
    :return: (list) of dictionary rows
    """

    row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
    list_of_rows.append(row) # Append new dictionary item to list
    return list_of_rows

```

Figure 2: Use of append() method to add data

Next a function was needed to remove data from the list of tasks from the file "Todofile.txt". A function was created using a for loop to iterate through the list of tasks, when a match is found based on the task, the user provides, the remove() method is used and the list minus the task is returned (figure 3).

```

@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
    """

    for row in list_of_rows: # Iterate through list
        if row["Task"].lower() == task.lower(): # Looking for a key that matches item entered by user
            list_of_rows.remove(row) # Remove both key and value from list
    return list_of_rows

```

Figure 3: Finding and removing a dictionary item

To save the new list that is created when either adding or removing an item the function "write_data_to_file" is used. This function uses the parameters of a file name and the modified list. The file is opened using the open() method with the "w" parameter followed by a for-loop to add each row of the new list to the text file. A print statement is used to inform the user their file has been saved and then closes the file using the close() method (figure 4).

```

@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
    """

    file_name = open(file_name, "w")
    for row in list_of_rows:
        file_name.write(row["Task"] + "," + row["Priority"] + "\n")
    file_name.close()
    print("Your tasks have been saved to the file ") # Let user know item has been added to file

    return list_of_rows

```

Figure 4: Opening and writing the new list to a text file

Input_new_task_and_priority was the next function where code was added. This function has a user enter a new task and priority. The function was created under the class IO and uses the input statement to allow a user to enter a new task and priority, and return the data that was just entered (figure 5).

```

@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 = (input("Enter a task: ")).strip() # Have user enter a new task
    priority = (input("Enter the priority: ")).strip() # Have user enter new priority
    return task, priority

```

Figure 5: Ask user to enter a new task and priority

Determining the task a user would want to have removed was handled just as like the entry of a new task was. A new function created under the IO class was created with the input function used asking the user to enter the task they would like to remove from the list of tasks and return the task that will be removed (figure 6).

```

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

    :return: (string) with task
    """

    taskRemove = input("Enter the task to remove: ")
    return taskRemove

```

Figure 6: User enters task to be removed

Finally all functions were nested under a while loop and controlled by conditional statements that were dependent on the choice the user made from the main menu, this piece of the script was provided by our instructor.

Validating the Script

Validating this script works as intended requires two approaches: (1) through Pycharm and (2) using the Shell Operator.

Through Pycharm:

First, I choose option 1 to add a new task and priority (figure 7). All tasks and priorities including the new task, are shown (figure 8). Next I choose 2 and remove "Sleep" (figure 9). Finally I save the data to a file and exit. Next I go to Finder >Documents >_PythonClass >Assignment06 >ToDoList.txt and open the file and verify a file has been created with the user input (figure 13).

Using the Shell Operator:

To validate the code works in the command shell I copy the path and paste it to the command line. I run the script using different data from what was used in PyCharm, a task of "clean", and capture the results of the code to demonstrate that the code works as intended (figure 11, figure 12). Finally, I use Finder again to locate the new text file created (TodoFile.txt) and validate the new data is written to the file (Figure 13).

```
***** The current tasks ToDo are: *****
Write Script (High)
Jump (Low)
Swim (Medium)
Sleep (High)
Run (High)
*****

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

Enter a task: Type
Enter the priority: Low
```

Figure 7: Choice 1 add new task and priority

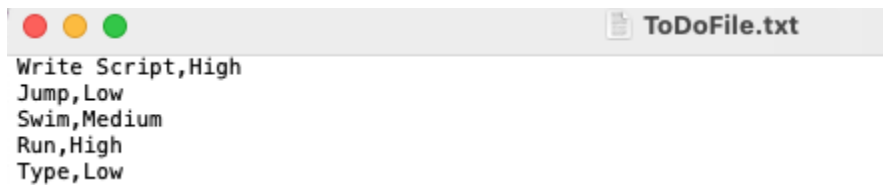
```
***** The current tasks ToDo are: *****
Write Script (High)
Jump (Low)
Swim (Medium)
Sleep (High)
Run (High)
Type (Low)
*****
```

Figure 8: “Type” added

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

Enter the task to remove: Sleep
***** The current tasks ToDo are: *****
Write Script (High)
Jump (Low)
Swim (Medium)
Run (High)
Type (Low)
*****
```

Figure 9: Display tasks with “Sleep” removed



The image shows a window titled 'ToDoFile.txt' with a list of tasks. The tasks are: Write Script,High; Jump,Low; Swim,Medium; Run,High; and Type,Low. The text is displayed in a monospaced font.

```
Write Script,High
Jump,Low
Swim,Medium
Run,High
Type,Low
```

Figure 10: File created with new task and “sleep” removed

```
Assignment_06 — Python Assignment_06.py — 126x18
(base) Amilas-MacBook-Pro:Assignment_06 amilapatel$ python3 Assignment_06.py
***** The current tasks ToDo are: *****
Write Script (High)
Jump (Low)
Swim (Medium)
Run (High)
Type (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] - █
```

Figure 11: Run code in shell

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

Enter a task: Clean
Enter the priority: High
***** The current tasks ToDo are: *****
Write Script (High)
Jump (Low)
Swim (Medium)
Run (High)
Type (Low)
Clean (High)
```

Figure 12: Adding a “Clean”

```
ToDoFile.txt
Write Script,High
Jump,Low
Swim,Medium
Run,High
Type,Low
Clean,High
```

Figure 13: Text file created

Summary

The use of classes and functions proved helpful in completing this assignment. Using functions allowed the script to be compartmentalized and then put together in the main block of code where each function could be called as needed.