Kyle Odland
May 25, 2020
Foundations of Programming: Python
Assignment 06
https://github.com/oddlandd/IntroToProg-Python

Using Functions

Introduction

The sixth assignment for Foundations of Programming: Python was to modify an existing script for a To-Do list, to add functionality to the program. The program works by loading an existing to-do list, displaying what is currently on the to-do list, then giving the user a menu of options - to add a new task, to remove a task, to save the to-do list to a file, to reload the to-do list data from the file, and to exit the program. The assignment was to write functions in the code that would perform these different actions.

Coding

The starter code given in the assignment provided a structure for how the program should operate. The main body of the program starts by loading data from a ToDo List text file into memory. This is accomplished with the Processor.read_data_from_file() function. The code for this function was included in the starter code. The code passes the file name and the existing list of rows into the function, and returns a new list of rows where each row is a dictionary containing a line of data from the file. The ToDo list text file is structured like Task,Priority, with a new line for each item, so the keys for the dictionary are “Task” and “Priority”.

After data is loaded from the file, the rest of the program is structured like a while loop with a series of if/elif statements, where the user chooses from a menu of options and each “if” statement executes the code for the user’s choice. The menu of options presented to the user are shown in Figure 1.

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

Figure 1: Menu of options for user

The code for each user choice was up to me to define. Choice 5 was included in the starter code, which just involved printing a goodbye message and breaking out of the while loop, ending the program.

Choice 4 for the user is to reload the data from the file, getting rid of whatever data exists in memory. When the user chooses this option, the starter code prompts them to choose yes or no, making sure they want to reload the data. If they say yes, the program calls the same Processor.read_data_from_file() function that was used at the beginning of the program. This overwrites the list of rows with whatever was in the text file. The code for this choice is shown in Listing 1. As given in the starter code, this portion of the code also calls the function IO.input_press_to_continue(), which displays the status of the data reload to the user, and prompts a user input to continue the program.

```
elif strChoice == '4':  # Reload Data from File
  print("Warning: Unsaved Data Will Be Lost!")
  strChoice = IO.input_yes_no_choice("Are you sure you want to reload data from file? (y/n) -  ")
  if strChoice.lower() == 'y':
    # read file data and overwrite lstTable
    lstTable, strStatus = Processor.read_data_from_file(strFileName, lstTable)
    IO.input_press_to_continue(strStatus)
  else:
    IO.input_press_to_continue("File Reload Cancelled!")
  continue  # to show the menu
```

Listing 1: Code for user choice 4, to reload the data from the file

My next step in writing the program was to write the code for user choice 3, saving the existing task/priority data back to the file. When the user chooses this option, they are prompted with a decision about whether to save. If they choose yes, I have the program call the Processor.write_data_to_file() function. The code for this function is very similar to what was done in Assignment05, with the difference being that it is accomplished in a function call, rather than in the main body of the script. Listing 2 shows the code for the Processor.write_data_to_file() function.

```
def write_data_to_file(file_name, list_of_rows):
  file = open(file_name, "w")
  for row in list_of_rows:
    file.write(row["Task"] + "," + row["Priority"] "\n")
  file.close()
  return list_of_rows, 'Success'
```

Listing 2: Code for Processor.write_data_to_file() function, to save the ToDo data to a file

My next step was to add functionality to user choice 1, adding a new task to the ToDo list. This is accomplished in the code by calling an IO function input_new_task_and_priority() and then

calling a Processor function add_data_to_list(). The input_new_task_and_priority function asks the user to input a new task and a priority level. These get passed into the add_data_to_list function, along with the existing table of data, which then appends a dictionary row onto the table of data and returns the result. Listing 3 shows the code for the input_new_task_and_priority() function and Listing 4 shows the code for the add_data_to_list() function.

```
def input_new_task_and_priority():
  task = input("Enter a new task: ")
  priority = input("Enter the priority of this task: ")
  return task, priority
```

Listing 3: Code for IO.input_new_task_and_priority() function, to get input for a new task

```
def add_data_to_list(task, priority, list_of_rows):
  row = {"Task": task.strip(), "Priority": priority.strip()}  # new dictionary row
  list_of_rows.append(row)
  return list_of_rows, 'Success'
```

Listing 4: Code for Processor.add_data_to_list() function, to add a new task to the list

The final step for the code was developing the functionality for user choice 2, to remove an existing task from the list. This is split into an IO function input_task_to_remove(), which asks the user to enter a task to remove, and a Processor function remove_data_from_list(). The code for removing data from list is very similar to what was done in Assignment05, and is shown in Listing 5. The code looks at each row of the table of data that is passed in, and sees if the task that is passed in is the same as the task in that row. If it is, it removes that row and returns “Success”. If it’s not, it returns a message that the task was not on the list.

```
def remove_data_from_list(task, list_of_rows):
  flag = False
  for row in list_of_rows:
    if row["Task"].lower() == task.lower():  # if the task in the dictionary matches the task to remove
      list_of_rows.remove(row)  # remove the row from the table
      flag = True  # switch the counter to show that an item was removed
      break
  if not flag:
    return list_of_rows, "Your task was not on the list!"
  else:
    return list_of_rows, 'Success'
```

Listing 5: Code for Processor.add_data_to_list() function, to add a new task to the list

Running the Code

With every user choice covered, I was ready to test out the final product. Figure 2 shows a sample of the program running in PyCharm.

```
/usr/local/bin/python3.8 /Users/Kyle/Documents/_Pythor
******* The current Tasks ToDo are: *******
Jog (4)
Cook (5)
mow the lawn (1)
*******************************************


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


Which option would you like to perform? [1 to 5] - 2

Enter a task to remove: mow the lawn
Success
Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Jog (4)
Cook (5)
*******************************************
```

Figure 2: Running the code for Assignment06 in PyCharm

Running the code in the terminal gives the output shown in Figure 3.

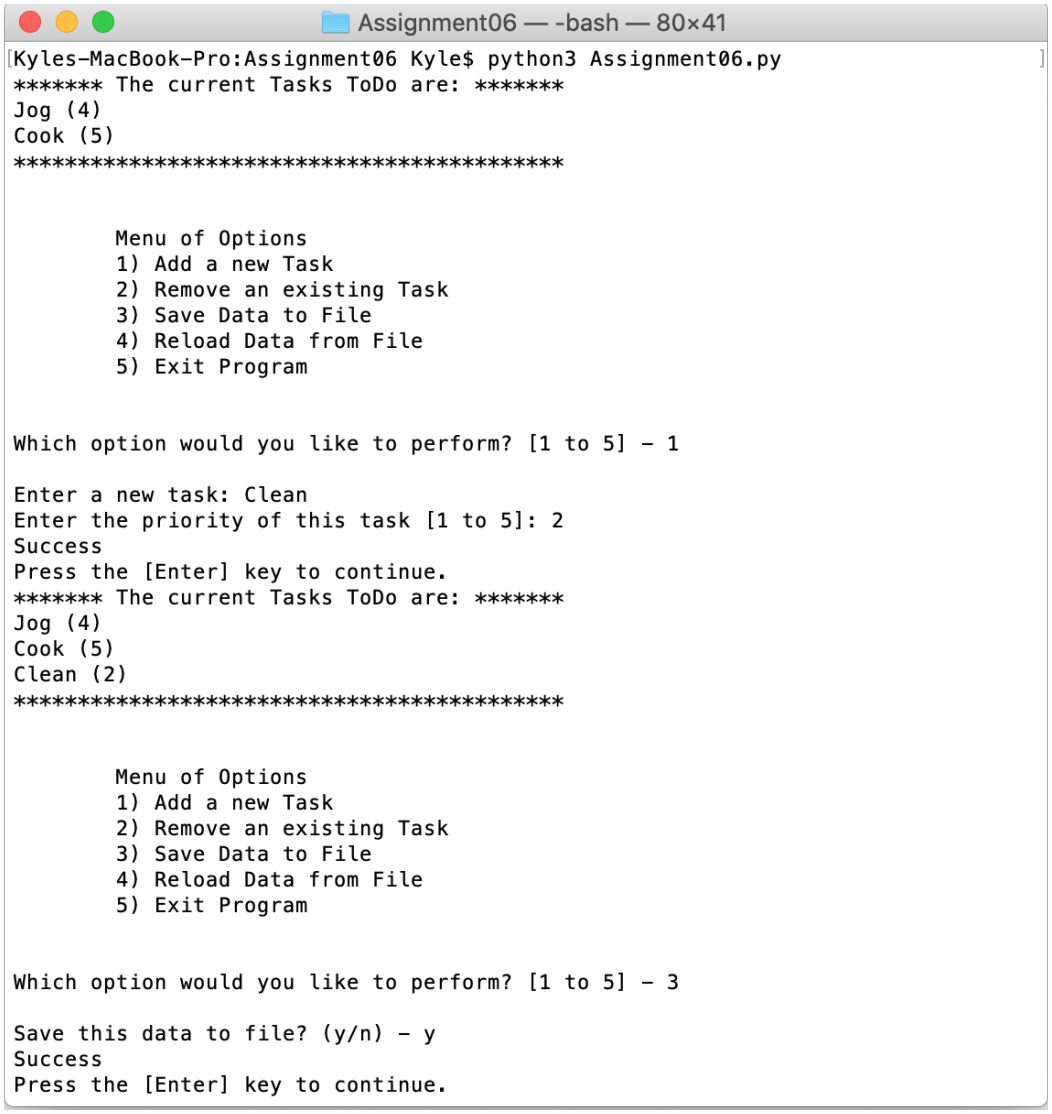

Figure 3: Running the code for Assignment05 on OSX Terminal

The code appears to work well in both PyCharm and the Terminal. The output file from after the two shown iterations of the program is shown in Figure 4.

Figure 4: Output text file with a task list

Summary

This module took the program that we wrote in Assignment05, and modified it using function calls to separate out the Processing section of the code from the Input/Output section of code. This organizes the script, making it easier to read and making it easier to add functionality. This method of using classes and functions to separate the different sections of what the program is doing seems like it will be very helpful in more complex programs.