Kegan Sanchez

November 23, 2020

Foundations of Programming (Python)

Assignment 06

<https://github.com/kegans/IntroToProg-Python-Mod06>

The To Do List Program 2.0

# Introduction

This week, I modified another Python script that manages a “To Do list” which contains two columns of data, “Task” and “Priority,” and loads the columns into a Python dictionary object. Each dictionary object represents one row of data, and the rows were added to a Python list object to create a table of data. I took my code that added functionality to process the data based on the user’s choice and incorporated it within functions in classes. This assignment included these new components as well as the PyCharm debugger and GitHub web pages.

# Modifying the Program

I began my program by creating a new project in PyCharm and adding the starter file to my project. I updated the changelog in the script’s header and started adding code to perform the assignment’s tasks.

## Adding Data

The first class in the starter file, called “Processor,” was listed just below the declared variables near the header of the script and included functions to perform the program’s processing tasks. One of the TODO functions requiring me to add code was called add\_data\_to\_list(). I included document string details and updated my variable names to match the parameters. (Figure 1).

@staticmethod

def add\_data\_to\_list(task, priority, list\_of\_rows):

# TODO: Done

""" Adds data specified by user input into a list of dictionary rows

:param task: (string) with name of task:

:param priority: (string) with level of priority:

:param list\_of\_rows: (list) to append with data:

:return: (list) of dictionary rows

"""

row = {"Task": task.lower(), "Priority": priority}

list\_of\_rows.append(row)

print("Item added.")

return list\_of\_rows, 'Success'

***Figure 1: Processing: Adding Data***

Since the add\_data\_to\_list() function did not handle user input, I searched for the IO class in the script which was intended to perform input and output tasks and located the function called input\_new\_task\_and\_priority(). I then added code that would prompt the user for a task name and priority (Figure 2).

@staticmethod

def input\_new\_task\_and\_priority():

# TODO: Done

""" Retrieve user input for new task and priority to add

:return: strings

"""

task = str(input("Task: "))

priority = str(input("Priority: "))

return task, priority

***Figure 2: Input/Output: Adding Data***

Since the IO function returned two variables, I used the appropriate gloabal variables, strTask and strPriority to capture them. I then passed these variables into the Processor function and captured the returned values in their associated global variables, lstTable and strStatus. The program was already set up to pass strStatus into the IO input\_press\_to\_continue() function, and then continue through the greater while() loop (Figure3).

if strChoice.strip() == '1': # Add a new Task

# TODO: Done

strTask, strPriority = IO.input\_new\_task\_and\_priority()

lstTable, strStatus = Processor.add\_data\_to\_list(strTask, strPriority, lstTable)

IO.input\_press\_to\_continue(strStatus)

continue # to show the menu

***Figure 3: Main Body: Adding Data***

## Removing Data

The next Processing class method was called remove\_data\_from\_list()and involved removing data from the list table of dictionary rows. I had already programed this functionality in the last assignment, so I copied that code, updated the variable names to match the given parameters, and included document string details (Figure 4).

@staticmethod

def remove\_data\_from\_list(task, list\_of\_rows):

# TODO: Done

""" Removes data specified by user input from a list of dictionary rows

:param task: (string) with name of task:

:param list\_of\_rows: (list) to append with data:

:return: (list) of dictionary rows, (string) status message

"""

taskFound = 0

for i in range(len(list\_of\_rows)):

if list\_of\_rows[i]["Task"] == task.lower():

del list\_of\_rows[i]

taskFound = 1

break

if taskFound:

print("Item deleted.")

else:

print("Item not found.")

return list\_of\_rows, 'Success'])

***Figure 4: Processing: Removing Data***

The IO operations were once again completed by a different function within the class performing input/output tasks. I located the associated function called input\_task\_to\_remove()and added my code and the appropriate document string details (Figure 5).

@staticmethod

def input\_task\_to\_remove():

# TODO: Done

""" Retrieve user input for task to remove

:return: string

"""

task = str(input("Which task would you like to delete? "))

return task

***Figure 5: Input/Output: Removing Data***

Finally, I brought it all together by capturing the return value of the IO function in strTask, passing it into the Processor function along with the global list table variable, lstTable, then capturing the return value of this function within the global list table variable and status variable, strStatus. The status variable was again passed into the IO input\_press\_to\_continue() function and the program continued through the while() loop (Figure 6).

elif strChoice == '2': # Remove an existing Task

# TODO: Done

strTask = IO.input\_task\_to\_remove()

lstTable, strStatus =Processor.remove\_data\_from\_list(strTask, lstTable)

IO.input\_press\_to\_continue(strStatus)

continue # to show the menu

***Figure 6: Main Body: Removing Data***

## Writing and Reloading Data

The final TODO items in the program involved saving and reloading data to/from a text file. I located the function within the Processing class that was intended to accomplish the writing operation, inserted my code, updated the parameter variable names, and included document string details (Figure 7).

@staticmethod

def write\_data\_to\_file(file\_name, list\_of\_rows):

# TODO: Done

""" Writes data stored in the list table to a text file

:param file\_name: (string) with name of text file:

:param list\_of\_rows: (list) to reference when writing data:

:return: (list) of dictionary rows, (string) status message

"""

objFile = open(file\_name, "w")

for row in list\_of\_rows:

objFile.write(row["Task"] + "," + row["Priority"] + "\n")

objFile.close()

print("File saved.")

return list\_of\_rows, 'Success'

***Figure 7: Processing: Writing Data***

In the main body of the program, I added functionality to call the function that would write data to a file and stored the return values of that function within the lstTable and strStatus variables. To reload the data in the final elif clause, I used the Processing function that already existed in the program called read\_data\_from\_file(), passed in the file name and list table, then once again captured the return values in lstTable and strStatus (Figure 8).

elif strChoice == '3': # Save Data to File

strChoice = IO.input\_yes\_no\_choice("Save this data to file? (y/n) - ")

if strChoice.lower() == "y":

# TODO: Done

lstTable, strStatus = Processor.write\_data\_to\_file(strFileName, lstTable)

IO.input\_press\_to\_continue(strStatus)

else:

IO.input\_press\_to\_continue("Save Cancelled!")

continue # to show the menu

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':

# TODO: Done

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

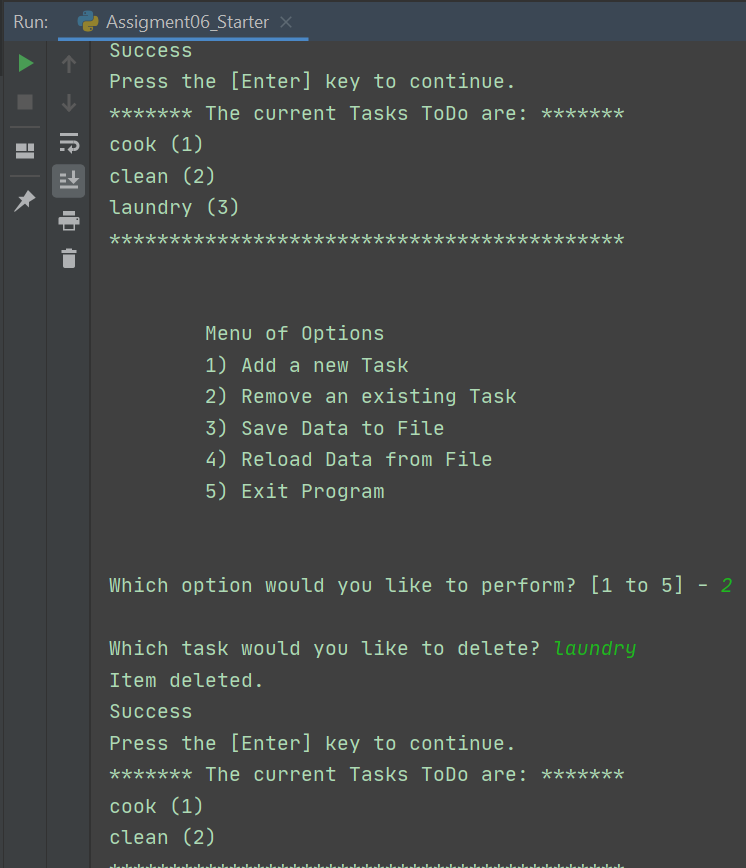
***Figure 8: Main Body: Writing and Reloading Data***

# Testing

When I began creating this program, I completed all coding and testing primarily in the PyCharm IDE. As part of this assignment, I also tested it in the command terminal.

## PyCharm

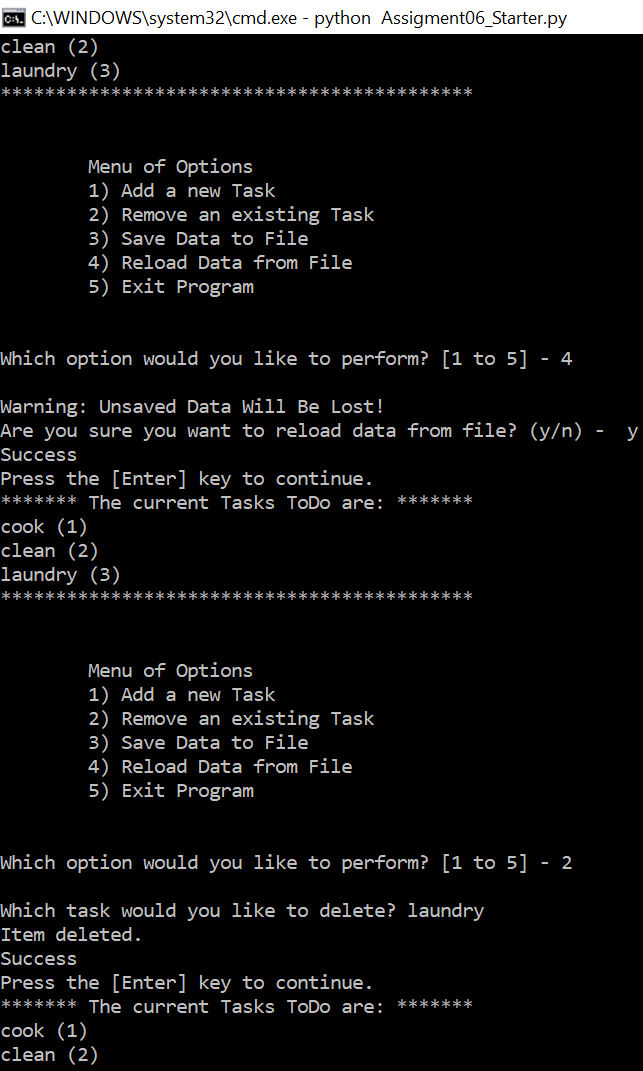
I first ran my To Do List 2.0 program in PyCharm and received the desired output, including displaying the status message (Figure 9).



***Figure 9: PyCharm Testing***

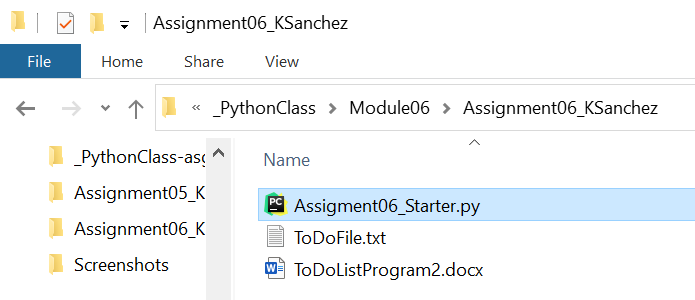
## Command Terminal

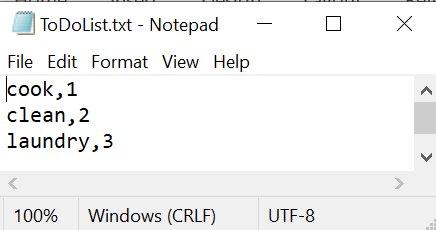
I also opened a command terminal, changed the directory to my assignment folder, and ran my Assignment06\_Starter.py script file. I received the desired output here, as well (Figure 10).



***Figure 10: Command Terminal Testing***

Finally, I checked that the text document was saved in my assignment folder, opened it, and confirmed that the data I entered was written into the file correctly (Figure 8).





***Figure 8: Verifying Data Saved to Text File***

# Summary

After modifying the code for each menu option and seeing my code run successfully, I reflected on what I had learned. I not only gained experience in how to add code within functions stored in classes, but also how to reference those functions, pass variables into them, and then capture their return values. I will keep these lessons in mind for future programs and continue looking for ways to keep using efficient operation handling by incorporating functions and classes that improve scalability.