Rex Sarabia

10 August 2022

IT FDN 110 B

Assignment06

GitHub: <https://github.com/rsar-uw/IntroToProg-Python-Mod06> (External)

Python Script: To Do List v1.0

Table of Contents

[2 Introduction 2](#_Toc111401769)

[3 My system information 2](#_Toc111401770)

[3.1 Operating system (OS) 3](#_Toc111401771)

[3.2 Console application 3](#_Toc111401772)

[3.3 Shell 3](#_Toc111401773)

[3.4 Python 4](#_Toc111401774)

[3.5 Integrated Development Environment (IDE) 4](#_Toc111401775)

[3.6 Directory / File path 5](#_Toc111401776)

[4 Module assignment 6](#_Toc111401777)

[4.1 Requirements 6](#_Toc111401778)

[4.1.1 Out of scope 6](#_Toc111401779)

[4.1.2 Lessons learned 6](#_Toc111401780)

[4.2 Design 7](#_Toc111401781)

[4.2.1 Standard elements 7](#_Toc111401782)

[4.2.1.1 Code style 7](#_Toc111401783)

[4.2.1.2 Exceptions 7](#_Toc111401784)

[4.2.1.3 Script header 7](#_Toc111401785)

[4.2.1.4 Program start 8](#_Toc111401786)

[4.2.1.5 Saving files 8](#_Toc111401787)

[4.2.2 Program architecture 8](#_Toc111401788)

[4.2.2.1 Start program, Show data from file 10](#_Toc111401789)

[4.2.2.2 Menu 12](#_Toc111401790)

[4.2.2.3 Menu option 1 – Add a new task 14](#_Toc111401791)

[4.2.2.4 Menu option 2 – Remove an existing task 16](#_Toc111401792)

[4.2.2.5 Menu option 3 – Save data to file 18](#_Toc111401793)

[4.2.2.6 Menu option 4 – Exit the program 20](#_Toc111401794)

[4.2.3 Proposed solution 21](#_Toc111401809)

[4.3 Test 27](#_Toc111401810)

[4.3.1 Procedure 27](#_Toc111401811)

[4.3.2 Results 32](#_Toc111401812)

[4.4 Execution 32](#_Toc111401813)

[4.4.1 Terminal 32](#_Toc111401814)

[4.4.2 Results 35](#_Toc111401815)

[5 Summary 35](#_Toc111401816)

[6 References 36](#_Toc111401817)

[6.1 Schema 36](#_Toc111401818)

[6.1.1 Books 36](#_Toc111401819)

[6.1.2 Websites 36](#_Toc111401820)

[6.2 Sources 36](#_Toc111401821)

# Introduction

My expectation for this assignment is to largely iterate on process (i.e., working off somebody else’s code, module-based development / integration) and documentation rather than from adding features to improve usability. Additionally, this week’s assignment expands on our use of GitHub, so I expect to allocate more time working with GitHub and adapting my workflow in addition to making general improvements to workflow efficiencies. To facilitate debugging and improve documentation, I expect to add more code as “temporary placeholders” for functioning code as the “integration” of individual modules will need to be tested as the main program code is updated with code developed from individual “modules". Specifically, I think it would be helpful to see how variable values are passed into / returned between functions.

Overall, I intend to keep my code as “similar” to the starter script as possible for the purpose of: (1) reduce time spent on troubleshooting optional features, (2) improve comparability with programs submitted by classmates.

[Table of Contents]

# My system information

I do not expect to include this information in every assignment. The purpose of including this information is to serve as a reference documentation for future needs (e.g., troubleshooting).

Where applicable, the information below will be updated after significant system component version updates, and the addition of new or replacement of existing system components (i.e., deltas from this information will be recorded).

[Table of Contents]

## Operating system (OS)

From the Apple menu, top left corner of screen:

 > About This Mac

**macOS Monterey version 12.4** (Figure 1)

****

Figure 1. Screen capture of OS version

[Table of Contents]

## Console application

**Terminal version 2.12.7 (445)**

*For more information, refer to A01-RSar.docx – Section 3.2.*

[Table of Contents]

## Shell

**zsh version 5.9**

*For more information, refer to A01-RSar.docx – Section 3.3.*

[Table of Contents]

## Python

**Python version 3.10.5**

*For more information, refer to A01-RSar.docx – Section 3.4.*

[Table of Contents]

## Integrated Development Environment (IDE)

Per instruction of Randal Root, using PyCharm Community Edition (CE) as default IDE for Module03. “Step 5.2 Create a new Project n PyCharm” (Randall, R. \_ Assignment03\_instructions.docx, Self-published, 2019).

Download and install PyCharm CE for macOS from JetBrains:

<https://www.jetbrains.com/pycharm/download/#section=mac>

Open PyCharm

From the PyCharm menu, top left corner of screen:

PyCharm > About PyCharm

**PyCharm version 2022.1.4 (Community Edition)** (Figure 2)



Figure 2. Screen capture of PyCharm version

[Table of Contents]

## Directory / File path

Open Terminal

Navigate to Assignment05 working folder.

Enter the following command (see Figure 3):

|  |
| --- |
| cd documents/\_pythonclass/module06/a06rsarabia |

Figure 3. Command in Terminal for navigating to assignment directory

This folder and its contents will be compressed into .zip file and submitted for Assignment06.

Enter the following command in Terminal to return the directory path (Figure 4):

|  |
| --- |
| pwd |

Figure 4. Command in Terminal to return directory path

Enter the following command in Terminal to return the directory content (see Figure 5):

|  |
| --- |
| ls -la |

Figure 5. Command in Terminal to return directory contents (visible and invisible files)

**/Users/rex/Documents/\_PythonClass/Module06/A06RSarabia** (Figure 6)

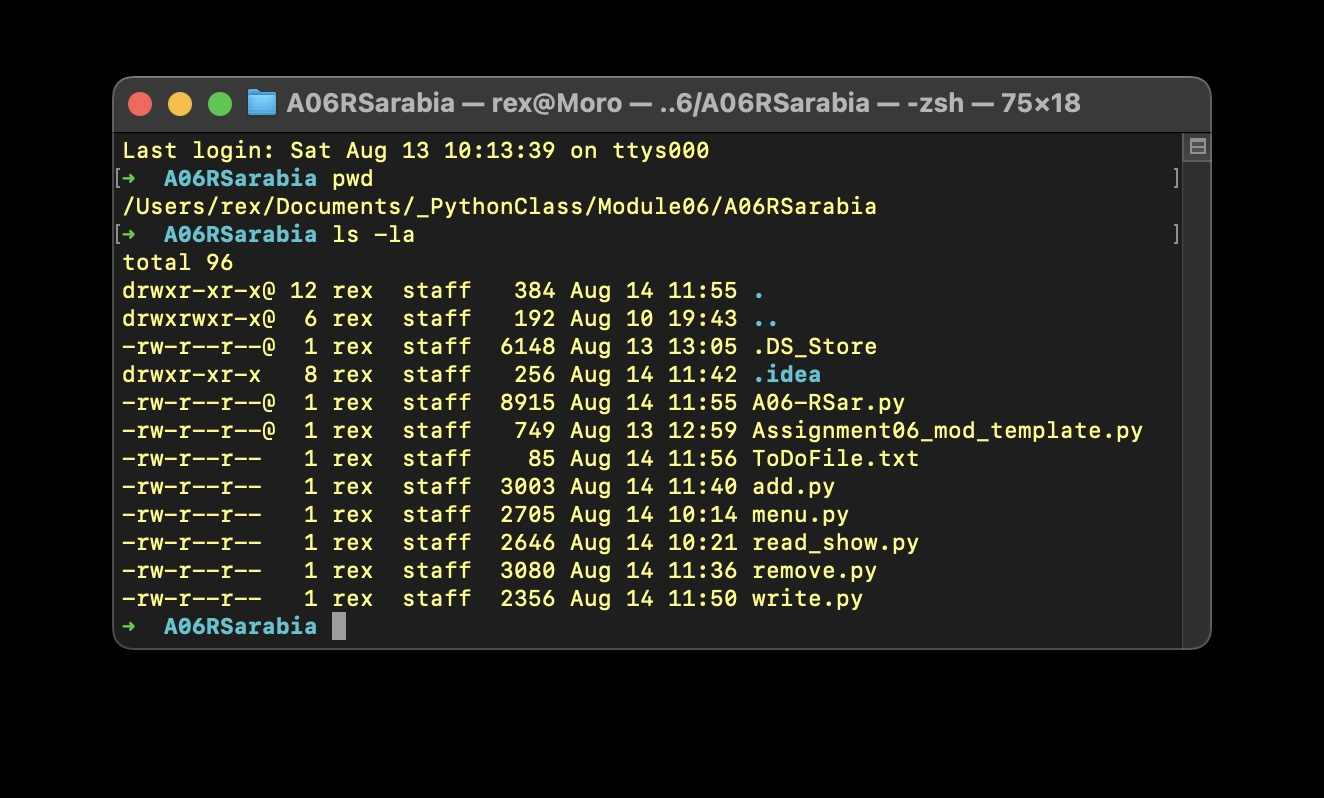


Figure 6. Screen capture of assignment directory and contents

[Table of Contents]

# Module assignment

## Requirements

Modify a script that manages a "ToDo list."

Modify and use for your program called “Assigment06\_Starter.py”.

Currently the code loads data from a file into a Python List of Dictionary objects. However, the code only uses a few functions, and your job is to add more functions to organize the code.

Add the starter file, "Assigment06\_Starter.py," to your project.

Add code to your script that performs the assignment’s tasks:

1. Step 1 – When the program starts, load data from ToDoFile.txt
2. Step 2 – Display a menu of choices to the user
3. Step 3 – Show current data
4. Step 4 – Process user’s menu choice
   1. Menu option 1 – Add a new task
   2. Menu option 2 – Remove an existing task
   3. Menu option 3 – Save data to file
   4. Menu option 4 – Exit the program

Additional details of program requirements included in corresponding write-up, see Section 4.2.2.x.

(Randall R., Assignment06\_instructions.docx, Self-published, 2022).

[Table of Contents]

### Out of scope

For the purposes of this document, documentation is limited to “Apply your knowledge” (Step 5) and “Document your knowledge” (Step 6) of the assignment. Additional assignment tasks related to “Post your Files to GitHub” (Step 7) through “Perform Peer Review” (Step 11) are not included in this write-up.

[Table of Contents]

### Lessons learned

As weeks progress and assignments are graded, I thought it would be helpful to keep a log of comments received from prior assignments (Figure 7). The simplest benefit is to serve as a log of lessons learned along the way and more practically, as immediate reminders to not repeat the same mistakes. *Steel sharpens steel.*

| **Assignment filename** | **Comment** | **Penalties** |
| --- | --- | --- |
| A01-RSar.docx | end the file with  print(‘(Press Enter to End Program)’) #Conclusion line  input() | -2 |
| A02-RSar.docx | any figure# in a caption below a images/code snippet is referenced by figure # in the text written up above | -1 |
| A03-RSar.docx | Filename should be: HomeInventory.py | -2 |
| A04-RSar.docx | the save/exit option didn't ask me if i wanted to save to the file  Note: see Figure 5 - Assignment04\_instructions.docx | -2 |
| A05-RSar.docx | Needed to post a link to your GitHub site on the assignment textbox | -1 |

Figure 7. Table of review comments from prior assignments

[Table of Contents]

## Design

This section includes the relevant components in the Python code that compose my proposed solution to the assignment.

[Table of Contents]

### Standard elements

#### Code style

By default, PyCharm will automatically perform checks of code style for PEP 8 (JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022) (External site). I made a conscious decision to follow the Problems/Solutions PyCharm identified, which in some cases conflict with the code style taught in class / prior assignments – honestly, it makes reading for errors in PyCharm a lot easier without having to mess with config settings of the feature.

For more information about PEP 8 code style guide: <https://peps.python.org/pep-0008/>

[Table of Contents]

#### Exceptions

Since I will be modifying “base code” provided, I have not changed the variable naming convention to keep code more comparable to the code submitted by classmates. In other circumstances, I will confirm to the code style specified in Section 4.2.1.1.

[Table of Contents]

#### Script header

In the script header (top of the script file), the following information must be recorded in the following format (Figure 8):

|  |
| --- |
| # ------------------------------- #  # Title: Assignment##  # Dev: RSar  # Desc: short description of script  # ChangeLog: (date,name,change)  # yyyy/mm/dd, Dev, Created File  # ------------------------------- # |

Figure 8. Source code for header

[Table of Contents]

#### Program start

Although not explicitly required, I’ve decided to begin all programs with the program title as the first line of display to the user. To ‘future-proof’ expansion on this feature, it will be declared as a variable and called in print() statement (Figure 9).

|  |
| --- |
| strProgramTitle = "To Do List XP v2.0" # Program name  print("\nWelcome to " + strProgramTitle + "!" # Display program name  "\n\n\tOpened file: '" + file\_name\_str + "'") |

Figure 9. Source code for program start

Since the base code for this program starts by defining variable for the data file (ToDoList.txt), I’ve modified the program start to include a message to the user that the data file has been opened. I manually created the text file to avoid errors if the file is opened in “read mode”.

[Table of Contents]

#### Saving files

“PyCharm automatically saves changes that you make in your files. Saving is triggered by various events, such as compiling, running, debugging, performing version control operations, closing a file or a project, or quitting the IDE.” (JetBrains, <https://www>.jetbrains.com/help/pycharm/saving-and-reverting-changes.html, 2022) (External site)

Per assignment requirements (Section 4.1), script filename and data output filename as follows:

* Script filename: **A06-RSar.py**
* Data output filename: **ToDoList.txt**
* Directory / file path: see Figure 4

[Table of Contents]

### Program architecture

I did not need to plan out the program holistically since the initial base code for this program had already been provided and we were explicitly required to work off that structure.

High-level workflow:

1. Copy “Assignment06\_Starter\_updated.py” script into Assignment06 script
2. Apply PEP8 formatting where practical
3. Create module template (Figure 10)
4. Create function tracking table ()
5. Create modules for related functions / tasks (e.g., menu options)
6. Iterate module
7. Test module
8. Integrate module to main Assignment06 script
9. Test Assignment06 script

Since the base code for this assignment is already prescribed, I created a template for further developing each menu option as its own module (Figure 10).

|  |
| --- |
| # ------------------------------- #  # Title: Assignment06  # Dev: RSar  # Desc: Module## - Module description  # ChangeLog: (date,name,change)  # 2022/mm/dd, RSar, Created module to complete Assignment  # ------------------------------- #  # Data ----------------------------------------------------------- #  # Declare variables and constants  # Processing ---------------------------------------------------- #  # Presentation (Input/Output) ----------------------------------- #  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load data from ToDoFile.txt.  # Step 2 - Display a menu of choices to the user  # Step 3 Show current data  # Step 4 - Process user's menu choice |

Figure 10. Pseudocode for module template

For this assignment, we are modifying existing custom defined functions. Therefore, to track development and status of modules, I created the table and updates / iterated as development progressed (Figure 11).

| **class** | **function** | **module** | **menu option** | **test, pre-integration** | **integrated** | **test, post-integration** |
| --- | --- | --- | --- | --- | --- | --- |
| Processor | read\_data\_from\_file | read\_show.py | menu loop | pass | yes | pass |
| IO | output\_current\_tasks\_in\_list | read\_show.py | menu loop | pass | yes | pass |
|  |  |  |  |  |  |  |
| IO | output\_menu\_tasks | menu.py | menu loop | pass | yes | pass |
| IO | input\_menu\_choice | menu.py | menu loop | pass | yes | pass |
|  |  |  |  |  |  |  |
| IO | input\_new\_task\_and\_priority | add.py | 1 | pass | yes | pass |
| Processor | add\_data\_to\_list | add.py | 1 | pass | yes | pass |
|  |  |  |  |  |  |  |
| IO | input\_task\_to\_remove | remove.py | 2 | pass | yes | pass |
| Processor | remove\_data\_from\_list | remove.py | 2 | pass | yes | pass |
|  |  |  |  |  |  |  |
| Processor | write\_data\_to\_file | write.py | 3 | pass | yes | pass |

Figure 11. Table for tracking module development status

[Table of Contents]

#### Start program, Show data from file

*Requirement 1: Load data from ToDoFile.txt When the program starts (Step 1)*

*Requirement 2: Show current data (Step 3)*

*Module: read\_show.py*

The program begins with declaring variables (including specifying the data file name), opens the data file: ToDoList.txt, and displays the information in file in a human-readable format (Figure 12). Also included in this module code is a print("\n\tfile\_name = " + file\_name) # temp\_debugging for debugging / troubleshooting purposes. This code and similar instances are expected to be commented-out for “production”.

|  |
| --- |
| # Data ----------------------------------------------------------- #  # Declare variables and constants  file\_name\_str = "ToDoFile.txt" # The name of the data file  row\_dic = {} # A row of data separated into elements of a dictionary  # {Task,Priority}  list\_of\_rows = [] # List of dictionary rows  table\_lst = [] # A list that acts as a 'table' of rows  choice\_str = "" # Captures the user option selection  # Processing ---------------------------------------------------- #  class Processor:  """ Performs Processing tasks """  @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  file = open(file\_name, "r")  for line in file:  task, priority = line.split(",")  row\_dic = {"Task": task.strip(), "Priority": priority.strip()}  list\_of\_rows.append(row\_dic)  file.close()  print("\n\tfile\_name = " + file\_name) # temp\_debugging  return list\_of\_rows  # Presentation (Input/Output) ----------------------------------- #  class IO:  # noinspection PyDecorator  @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("\n\t\*\*\*\*\*\*\* The current tasks To Do are: \*\*\*\*\*\*\*")  for row in list\_of\_rows:  print("\t" + row["Task"] + " (" + row["Priority"] + ")")  print("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load data from ToDoFile.txt.  Processor.read\_data\_from\_file(file\_name=file\_name\_str,  list\_of\_rows=table\_lst) # Read file  # Step 2 - Display a menu of choices to the user  # Step 3 Show current data  IO.output\_current\_tasks\_in\_list(list\_of\_rows=table\_lst) # Show \  # current data in the list/table  IO.output\_menu\_tasks() # Shows menu  choice\_str = IO.input\_menu\_choice() # Get menu option  # Step 4 - Process user's menu choice |

Figure 12. Source code for menu function

To simplify the code and in keeping with the intentions of this assignment, I did not carryover / add functions that were not already specified in the starter file (e.g., condition if file exists, row count).

[Table of Contents]

#### Menu

*Requirement 3: Display a menu of choices to the user (Step 2)*

*Module: menu.py*

Similar to Assignment05, Menu (and majority of the program is inside a while True: loop (Figure 13) where the user “quits” the program by triggering the break statement to exit the Menu loop. Additionally, I added “debugging” / dummy code to mimic the functions that would be called based on the selection of the user so that I can develop / test code in this module independently from the others. Unlike the Menu in the prior assignment, this program converts the menu options (“1-4”) to string datatype value and removes inadvertent spaces that may have been entered by the user: str(input("Which option would you like to perform? [1 to 4] - ")).strip().

|  |
| --- |
| # Data ----------------------------------------------------------- #  # Declare variables and constants  choice\_str = "" # Captures the user option selection  # Processing ---------------------------------------------------- #  # Presentation (Input/Output) ----------------------------------- #  class IO:  """ Performs Input and Output tasks """  @staticmethod  def output\_menu\_tasks():  """ Display a menu of choices to the user  :return: nothing  """  print("""  \tMenu of Options  \t1) Add a new Task  \t2) Remove an existing Task  \t3) Save Data to File  \t4) Exit Program  """)  @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()  return choice  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load data from ToDoFile.txt.  # Step 2 - Display a menu of choices to the user  while True:  # Step 3 Show current data  print("""  \tstart Menu loop  \tCall 1: \t\tIO.output\_current\_tasks\_in\_list()  \t# Show current data in the list/table")""") # temp\_debugging  IO.output\_menu\_tasks() # Shows menu  choice\_str = IO.input\_menu\_choice() # Get menu option  # Step 4 - Process user's menu choice  if choice\_str.strip() == '1': # Add a new Task  print("""  \tUser selected: \tOption 1 - 'Add a new task'  \tCall 1: \t\tinput\_new\_task\_and\_priority  \tCall 2: \t\tadd\_data\_to\_list""") # temp\_debugging  continue # to show the menu  elif choice\_str == '2': # Remove an existing Task  print("""  \tUser selected: \tOption 2 - 'Remove an existing task'  \tCall 1: \t\tinput\_task\_to\_remove  \tCall 2: \t\tremove\_data\_from\_list""") # temp\_debugging  continue # to show the menu  elif choice\_str == '3': # Save Data to File  print("""  \tUser selected: \tOption 3 - 'Save Data to File'  \tCall 1: \t\tinput\_task\_to\_remove  \tCall 2: \t\twrite\_data\_to\_file""") # temp\_debugging  print("\n\tData Saved!")  continue # to show the menu  elif choice\_str == '4': # Exit Program  print("\n\tUser selected: Option 4 - 'Exit program'")  # temp\_debugging  print("\n\tGoodbye!")  input("\n[Press ENTER key to quit.]")  break # by exiting loop |

Figure 13. Source code for displaying menu options to user and prompt for user instruction

[Table of Contents]

#### Menu option 1 – Add a new task

*Requirement 4: Process user’s menu choice (Step 4) – Add a new task (Menu option 1)*

*Module: add.py*

Unlike the corresponding Add function in Assignment05, this program will always accept all inputs from the user (Figure 14). Therefore, the user cannot return to the Menu without adding data; even Null values are captured and added to list table\_lst.

The general data flow: data from file > added to table\_lst (global) > copied to list\_of\_rows (local) > data is added to list\_of\_rows (local) > list\_of\_rows (local) data is copied to table\_lst (global).

To improve development efficiency, placed the main body code in while True loop so I don’t need to manually restart the module if I want to try adding more than one value at a time.

Note: The function will accept duplicate key values. Therefore, if one of the duplicate key values are removed by the user, the first “record” of a matching key value will be removed.

|  |
| --- |
| # Data ----------------------------------------------------------- #  # Declare variables and constants  row\_dic = {} # A row of data separated into elements of a dictionary \  # {Task,Priority}  list\_of\_rows = [] # List of dictionary rows  table\_lst = [] # A list that acts as a 'table' of rows  # Processing ---------------------------------------------------- #  class Processor:  """ Performs Processing tasks """  # noinspection PyDecorator  @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\_dic = {"Task": str(task).strip(),  "Priority": str(priority).strip()}  # TODO: Add Code Here!  list\_of\_rows.append(row\_dic)  print("\n\tAdded task: '" + task + " (" + priority + ")'")  print("\n\tProcessor.add\_data\_to\_list(list\_of\_rows) = " +  str(list\_of\_rows)) # temp\_debugging  return list\_of\_rows  # Presentation (Input/Output) ----------------------------------- #  class IO:  """ Performs Input and Output tasks """  # noinspection PyDecorator  @staticmethod  def input\_new\_task\_and\_priority():  """ Gets task and priority values to be added to the list  :return: (task, priority) with task and priority  """  task = str(input("\nWhat is the task? - "))  priority = str(input("What is the priority? - "))  print("\n\tIO.input\_new\_task\_and\_priority(task) = " + task) # \  # temp\_debugging  print("\tIO.input\_new\_task\_and\_priority(priority) = " +  priority) # temp\_debugging  return task, priority # TODO: Add Code Here!  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load data from ToDoFile.txt.  # Step 2 - Display a menu of choices to the user  # Step 3 Show current data  # Step 4 - Process user's menu choice  # Add a new Task  print("\n\tUser selected: \tOption 1 - 'Add a new task'")  # temp\_debugging  while True: # temp\_debugging  print("\n\tCall 1: \t\tIO.input\_new\_task\_and\_priority()")  # temp\_debugging  task, priority = IO.input\_new\_task\_and\_priority()  print("\n\tCall 2: \t\tProcessor.add\_data\_to\_list()")  # temp\_debugging  table\_lst = Processor.add\_data\_to\_list(task=task,  priority=priority,  list\_of\_rows=table\_lst)  print("\n\ttable\_lst = " +  str(table\_lst)) # temp\_debugging |

Figure 14. Source code to display list table contents to user

[Table of Contents]

#### Menu option 2 – Remove an existing task

*Requirement 5: Process user’s menu choice (Step 4) – Remove an existing task (Menu option 2)*

*Module: remove.py*

For this module to function without opening a data, I hardcoded simulated values (task and priority) for the list table\_lst (Figure 15).

The data flow for this module is similar to that of the add.py module: data from file > added to table\_lst (global) > copied to list\_of\_rows (local) > data is removed from list\_of\_rows (local) > list\_of\_rows (local) data is copied to “overwrites” table\_lst (global).

For the module to “search” for the task to be removed, it converts the user input to all lowercase letters and compares it against the converted lowercase key value: for row\_dic in list\_of\_rows: if row\_dic["Task"].lower() == task.lower().

To improve development efficiency, placed the main body code in while True loop so I don’t need to manually restart the module if I want to try adding more than one value at a time.

Note: If there are duplicate key values, then this function will remove the first “record” in the list list\_of\_rows / table\_lst.

|  |
| --- |
| # Data ----------------------------------------------------------- #  # Declare variables and constants  row\_dic = {} # A row of data separated into elements of a dictionary  list\_of\_rows = [] # List of dictionary rows  table\_lst = [{'Task': 'task1', 'Priority': 'p1'},  {'Task': 'task2', 'Priority': 'p2'},  {'Task': 'task3', 'Priority': 'p3'},  {'Task': 'task4', 'Priority': 'p4'}] # A list that \  # acts as a 'table' of rows {Task,Priority} - pre-populated values \  # for temp\_debugging  # Processing ---------------------------------------------------- #  class Processor:  """ Performs Processing tasks """  @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 row\_dic in list\_of\_rows: # Remove task  if row\_dic["Task"].lower() == task.lower():  print("\n\tRemoved task: '" + row\_dic["Task"] +  "(" + row\_dic["Priority"] + ")'")  list\_of\_rows.remove(row\_dic)  print("\n\tlist\_of\_rows = " + str(list\_of\_rows))  # temp\_debugging  print("\n\ttable\_lst = " + str(table\_lst)) # temp\_debugging  return list\_of\_rows  # Presentation (Input/Output) ----------------------------------- #  class IO:  """ Performs Input and Output tasks """  @staticmethod  def input\_task\_to\_remove():  """ Gets the task name to be removed from the list  :return: (string) with task  """  task = str(input("\nTask to remove?: ")).strip()  print("\n\tIO.input\_task\_to\_remove(task) = " + task)  # temp\_debugging  return task  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load data from ToDoFile.txt.  # Step 2 - Display a menu of choices to the user  # Step 3 Show current data  # Step 4 - Process user's menu choice  print("\n\tUser selected: \tOption 2 - 'Remove an existing task'")  # temp\_debugging  while True: # temp\_debugging  print("\n\tlist\_of\_rows = " + str(list\_of\_rows))  # temp\_debugging  print("\n\ttable\_lst = " + str(table\_lst)) # temp\_debugging  print("\n\tCall 1: \t\tProcessor.input\_task\_to\_remove")  # temp\_debugging  task = IO.input\_task\_to\_remove()  print("\n\tCall 2: \t\tProcessor.remove\_data\_from\_list()")  # temp\_debugging  table\_lst = Processor.remove\_data\_from\_list(task=task,  list\_of\_rows=  table\_lst)  print("\n\t\\end Option 2 loop.") # temp\_debugging |

Figure 15. Source code to write list table data to text file and quit the program

[Table of Contents]

#### Menu option 3 – Save data to file

*Requirement 6: Process user’s menu choice (Step 4) – Save data to file (Menu option 3)*

*Module: write.py*

For this module to function without opening a data, I hardcoded simulated values (task and priority) for the list table\_lst (Figure 16). When this function is called, it “copies” the values from the list table\_lst to the list list\_of\_rows. The list list\_of\_rows only contains data inside of the function, at the global level, it is empty: list\_of\_rows = [].

|  |
| --- |
| # 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}  list\_of\_rows = []  table\_lst = [{'Task': 'write\_test\_task1', 'Priority': 'p1'},  {'Task': 'write\_test\_task2', 'Priority': 'p2'},  {'Task': 'write\_test\_task3', 'Priority': 'p3'},  {'Task': 'write\_test\_task4', 'Priority': 'p4'}] # A \  # list that acts as a 'table' of rows - pre-populated values for  # Processing ---------------------------------------------------- #  class Processor:  """ Performs Processing tasks """  @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!  file\_obj = open(file\_name, "w")  for row\_dic in list\_of\_rows:  file\_obj.write(str(row\_dic["Task"]) + "," +  str(row\_dic["Priority"]) + "\n")  file\_obj.close()  print("\n\tlist\_of\_rows = " + str(list\_of\_rows)) # temp\_debugging  print("\n\tSaved to file: " + file\_name)  return list\_of\_rows  # Presentation (Input/Output) ----------------------------------- #  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load data from ToDoFile.txt.  # Step 2 - Display a menu of choices to the user  # Step 3 Show current data  # Step 4 - Process user's menu choice  print("\n\tUser selected: \tOption 3 - 'Save Data to File'"  "\n\tCall 1: \t\tProcessor.write\_data\_to\_file()") # temp\_debugging  table\_lst = Processor.write\_data\_to\_file(file\_name=  file\_name\_str,  list\_of\_rows=table\_lst) |

Figure 16. Source code to write the data to a file

[Table of Contents]

#### Menu option 4 – Exit the program

*Requirement 7: Process user’s menu choice (Step 4) – Exit the program (Menu option 4)*

*Module: main*

Similar to Assignment05, the program quits by breaking the menu loop (Figure 17). I added debug coding to maintain consistency with other menu options and modified the greeting for uniqueness.

|  |
| --- |
| # Data ----------------------------------------------------------- #  # Declare variables and constants  # Processing ---------------------------------------------------- #  # Presentation (Input/Output) ----------------------------------- #  # Main Body of Script ------------------------------------------- #  # Step 1 - When the program starts, Load data from ToDoFile.txt.  # Step 2 - Display a menu of choices to the user  while True:  # Step 3 Show current data  IO.output\_current\_tasks\_in\_list(list\_of\_rows=table\_lst) # Show \  # current data in the list/table  IO.output\_menu\_tasks() # Shows menu  choice\_str = IO.input\_menu\_choice() # Get menu option  # Step 4 - Process user's menu choice  elif choice\_str == '4': # Exit Program  # debug  print("\t\\\\Global (choice\_str) = " + choice\_str)  # /debug  print("\n\tByeeee!")  input("\n[Press ENTER key to quit.]")  break # exit Menu loop |

Figure 17. Source code to Exit the program

[Table of Contents]



### Proposed solution

The following source code is my program for Assignment06 (Figure 18).

|  |
| --- |
| # ------------------------------- #  # Title: Assignment06  # Dev: RSar  # Desc: Working with functions in a class,  # When the program starts, load each "row" of data  # in "ToDoToDoList.txt" into a python Dictionary.  # Add each dictionary "row" to a python list "table"  # ChangeLog: (date,name,change)  # 2022/01/01, RRoot, Created starter script  # 2022/08/13, RSar, Modified code to complete Assignment06  # 2022/08/14, RSar, Assembled code from modules  # 2022/08/16, RSar, Removed 'shadow global variable' error, Comment out debug  # code.  # ------------------------------- #  # Data ----------------------------------------------------------- #  # Declare variables and constants  strProgramTitle = "To Do List XP v2.0" # Program name  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  # list\_of\_rows = [] # List of dictionary rows  # Processing ---------------------------------------------------- #  class Processor:  """ Performs Processing tasks """  @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  file = open(file\_name, "r")  for line in file:  t, p = line.split(",")  row\_dic = {"Task": t.strip(), "Priority": p.strip()}  list\_of\_rows.append(row\_dic)  file.close()  # # debug  # print("\n\t\\\\file\_name = " + file\_name)  # # /debug  return list\_of\_rows  @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  """  # # debug  # print("\t\\\\Processor.add\_data\_to\_list(task) = " + task +  # "\n\t\\\\Processor.add\_data\_to\_list(priority) = " + priority)  # # /debug  row\_dic = {"Task": str(task).strip(),  "Priority": str(priority).strip()}  # TODO: Add Code Here!  list\_of\_rows.append(row\_dic)  # # debug  # print("\n\t\\\\Processor.add\_data\_to\_list(list\_of\_rows) = " +  # str(list\_of\_rows))  # # /debug  print("\n\tAdded task (priority): '" + task + " (" + priority + ")'")  return list\_of\_rows  @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 row\_dic in list\_of\_rows: # Remove task  if row\_dic["Task"].lower() == task.lower():  list\_of\_rows.remove(row\_dic)  # # debug  # print("\t\\\\Processor.remove\_data\_from\_list(task) = " +  # task.lower())  # # /debug  print("\n\tRemoved task (priority): '" + row\_dic["Task"] +  " (" + row\_dic["Priority"] + ")'")  # # debug  # print("\n\t\\\\Processor.remove\_data\_from\_list(list\_of\_rows) = " +  # str(list\_of\_rows))  # # /debug  return list\_of\_rows  @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!  file\_obj = open(file\_name, "w")  for row\_dic in list\_of\_rows:  file\_obj.write(str(row\_dic["Task"]) + "," +  str(row\_dic["Priority"]) + "\n")  file\_obj.close()  # # debug  # print("\n\t\\\\Processor.write\_data\_to\_file(list\_of\_rows)" +  # str(list\_of\_rows))  # # /debug  print("\n\tSaved data to file: " + file\_name)  return list\_of\_rows  # Presentation (Input/Output) ----------------------------------- #  class IO:  """ Performs Input and Output tasks """  @staticmethod  def output\_menu\_tasks():  """ Display a menu of choices to the user  :return: nothing  """  print("""  \tMenu of Options  \t1) Add a new Task  \t2) Remove an existing Task  \t3) Save Data to File  \t4) Exit Program  """)  @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()  # # debug  # print("\n\t\\\\IO.input\_menu\_choice(choice) = " + choice)  # # /debug  return choice  @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("\n\t\*\*\*\*\*\*\* The current tasks To Do are: \*\*\*\*\*\*\*")  for row in list\_of\_rows:  print("\t" + row["Task"] + " (" + row["Priority"] + ")")  print("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  @staticmethod  def input\_new\_task\_and\_priority():  """ Gets task and priority values to be added to the list  :return: (task, priority) with task and priority  """  task = str(input("\nWhat is the task? - "))  priority = str(input("What is the priority? - "))  # # debug  # print("\n\t\\\\IO.input\_new\_task\_and\_priority(task) = " + task +  # "\n\t\\\\IO.input\_new\_task\_and\_priority(priority) = " +  # priority)  # # /debug  return task, priority # TODO: Add Code Here!  @staticmethod  def input\_task\_to\_remove():  """ Gets the task name to be removed from the list  :return: (string) with task  """  task = str(input("\nTask to remove?: ")).strip()  # # debug  # print("\n\t\\\\IO.input\_task\_to\_remove(task) = " + task)  # # /debug  return task  # Main Body of Script ------------------------------------------- #  print("\nWelcome to " + strProgramTitle + "!" # Display program name  "\n\n\tOpened file: '" + file\_name\_str + "'")  # Step 1 - When the program starts, Load data from ToDoFile.txt.  Processor.read\_data\_from\_file(file\_name=file\_name\_str,  list\_of\_rows=table\_lst) # Read file data  # Step 2 - Display a menu of choices to the user  while True:  # # debug  # print("\n\t\\\\start Menu loop "  # "\n\t\\\\Call: \tIO.output\_current\_tasks\_in\_list()")  # # /debug  # Step 3 Show current data  IO.output\_current\_tasks\_in\_list(list\_of\_rows=table\_lst) # Show \  # current data in the list/table  # # debug  # print("\n\t\\\\Call: \tIO.output\_menu\_tasks()")  # # /debug  IO.output\_menu\_tasks() # Shows menu  # # debug  # print("\t\\\\Global (choice\_str) = " + choice\_str +  # "\n\t\\\\Call: \tIO.input\_menu\_choice()")  # # /debug  choice\_str = IO.input\_menu\_choice() # Get menu option  # Step 4 - Process user's menu choice  if choice\_str.strip() == '1': # Add a new Task  # # debug  # print("\t\\\\Global (choice\_str) = " + choice\_str +  # "\n\n\t\\\\Global (table\_lst) = " + str(table\_lst) +  # "\n\t\\\\Call: \tIO.input\_new\_task\_and\_priority()")  # # /debug  \_task, \_priority = IO.input\_new\_task\_and\_priority()  # # debug  # print("\t\\\\Call: \tProcessor.add\_data\_to\_list()")  # # /debug  table\_lst = Processor.add\_data\_to\_list(task=\_task,  priority=\_priority,  list\_of\_rows=table\_lst)  # # debug  # "\n\n\t\\\\Global (table\_lst) = " + str(table\_lst))  # # /debug  continue # to show the menu  elif choice\_str == '2': # Remove an existing Task  # # debug  # print("\t\\\\Global (choice\_str) = " + choice\_str +  # "\n\n\t\\\\Global (table\_lst) = " + str(table\_lst) +  # "\n\n\t\\\\Call: \tIO.input\_task\_to\_remove()")  # # /debug  \_task = IO.input\_task\_to\_remove()  # # debug  # print("\t\\\\Call: \tProcessor.remove\_data\_from\_list()")  # # /debug  table\_lst = Processor.remove\_data\_from\_list(task=\_task,  list\_of\_rows=table\_lst)  # # debug  # "\n\n\t\\\\Global (table\_lst) = " + str(table\_lst))  # # /debug  continue # to show the menu  elif choice\_str == '3': # Save Data to File  # # debug  # print("\n\t\\\\Global (choice\_str) = " + choice\_str +  # "\n\n\t\\\\Global (table\_lst) = " + str(table\_lst) +  # "\n\n\t\\\\Call: \tProcessor.write\_data\_to\_file()")  # # /debug  table\_lst = Processor.write\_data\_to\_file(file\_name=file\_name\_str,  list\_of\_rows=table\_lst)  # # debug  # print("\n\t\\\\Global (table\_lst) = " + str(table\_lst))  # # /debug  continue # to show the menu  elif choice\_str == '4': # Exit Program  # # debug  # print("\t\\\\Global (choice\_str) = " + choice\_str)  # # /debug  print("\n\tByeeee!")  input("\n[Press ENTER key to quit.]")  break # exit Menu loop |

Figure 18. Source code for my proposed solution to Assignment05

[Table of Contents]

## Test

### Procedure

For the purpose of this assignment, testing is performed in PyCharm IDE.

Open PyCharm

PyCharm > File > Open > directory path (Section 3.6) > A06-RSar.py

PyCharm > Run > “A06-RSar”

For this assignment and based on the limitations placed on the inputs of the user, I intend to limit my test cases to expected errors and a few valid input types (Figure 28).

| **Test flow ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
| 1 | Start program | Start program / Open data file and display contents | Pass |
| 2 | Flow: Add tasks to Save | Enter data  *Task: “task4”, Priority: “p4”*  *Task: “TASK6”, Priority: “P6”*    Save file    View output data file | Pass |
| 3 | Flow: Remove task to Save | Remove data  *Task: “TASK4”*    Save file    View output data file | Pass |

Figure 28. Summary of tests performed and results in PyCharm IDE

[Table of Contents]

### Results

Results from test cases all passed as the actual result matched what I was expecting for each of the value combinations input.

[Table of Contents]

## Execution

For the purpose of this assignment, execution is done via the Terminal which simulates executing in Production environment (PROD) as running the program in PyCharm IDE simulates testing environment (TEST).

[Table of Contents]

### Terminal

Open Terminal

Enter the following command (Figure 29):

|  |
| --- |
| python3 [file path][file name] |

Figure 29. Command for executing script in Terminal

Where:

file path: Figure 6

file name: Section 4.2.1.5

Re-perform test procedures (see Figure 30). For this assignment, the data inputted into HomeInventory.txt file has been erased and so the file is null.

| **Test flow ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
| 1 | Start program | Start program / Open data file and display contents | Pass |
| 2 | Flow: Add tasks to Save | Enter data  *Task: “task4”, Priority: “p4”*  *Task: “TASK6”, Priority: “P6”*    Save file    View output data file | Pass |
| 3 | Flow: Remove task to Save | Remove data  *Task: “TASK4”*    Save file    View output data file | Pass |

Figure 30. Summary of tests performed and results in Terminal

[Table of Contents]

### Results

Using the same input values per Section 4.3.1, produced the same results as testing for both error messages of invalid input as well as correctly formatted and stored outputs of valid user inputs.

[Table of Contents]

# Summary

In retrospect, I may have gone overboard on “debugging code” as the code of my proposed solution is doubled in length. However, I think that taking the additional time / effort in adding debugging code was helpful for me to see how data moves through the program (i.e., ETL) as well as how global vs local variables change throughout the program (e.g., table\_lst <-> list\_of\_rows, choice\_str <-> choice)

I would not expect to need this extent of “debugging code” for future assignments but considering this is the first assignment to make extensive use of defining custom functions with parameters / attributes, it is good to have done this at least once for reference. After taking this approach towards developing code in modules and focus on debugging, this experience has been especially helpful for appreciating: (1) coding consistency, (2) timing / placement of code.

Future discussion topics:

* Workflow (e.g., GitHub) best practices – continuous integration / automation
* Built-in / automated logging functions vs. manually coding “log” functions

[Table of Contents]

# References

## Schema

### Books

“Quoted text“ (Author Last name Author First name initial., Title of book, Publisher, Year published)

### Websites

“Quoted text“ (Website name, URL, yearAccessed) (External site)

[Table of Contents]

## Sources

*Note: It is assumed that knowledge builds, therefore, duplicate sources already included in prior assignments have been removed – unless it has been directly referenced within this assignment.*

Sarabia R., A01-RSar.docx, Self-published, 2022

Sarabia R., A02-RSar.docx, Self-published, 2022

Sarabia R., A03-RSar.docx, Self-published, 2022

Sarabia R., A04-RSar.docx, Self-published, 2022

Sarabia R., A05-RSar.docx, Self-published, 2022

JetBrains, <https://www.jetbrains.com/help/pycharm/saving-and-reverting-changes.html>, 2022 (External site)

JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022 (External site)

Randall, R. \_ Assignment06\_instructions.docx, Self-published, 2019

Randall R., \_Mod6PythonProgrammingNotes.docx, Self-published, 2019

Randall R., Assigment06\_Starter\_updated.py, Self-published, 2022

Randall R., Assignment06-Updated-Answer.pdf, Self-published, 2022

[Table of Contents]