Kyle O’Donnell
August 10, 2020
Foundations of Programming: Python
Assignment05

Creating the “To Do” List Program in Python

Introduction

This paper documents the process of creating the ‘ToDoList.py’ program in completion of Assignment 5 for Randal Root’s course, Foundations of Programming: Python. The program allows users to view a ‘to do’ list by reading data from a text file, add/remove tasks from the list, and save the modified list back to the text file. The steps covered here include adapting the script from a template, testing the program in PyCharm and Terminal, and confirming the script successfully saved data to a file. The process incorporates several concepts covered in week 5’s lesson, such as dictionaries, “Separation of Concerns,” and Try-Except error handling.

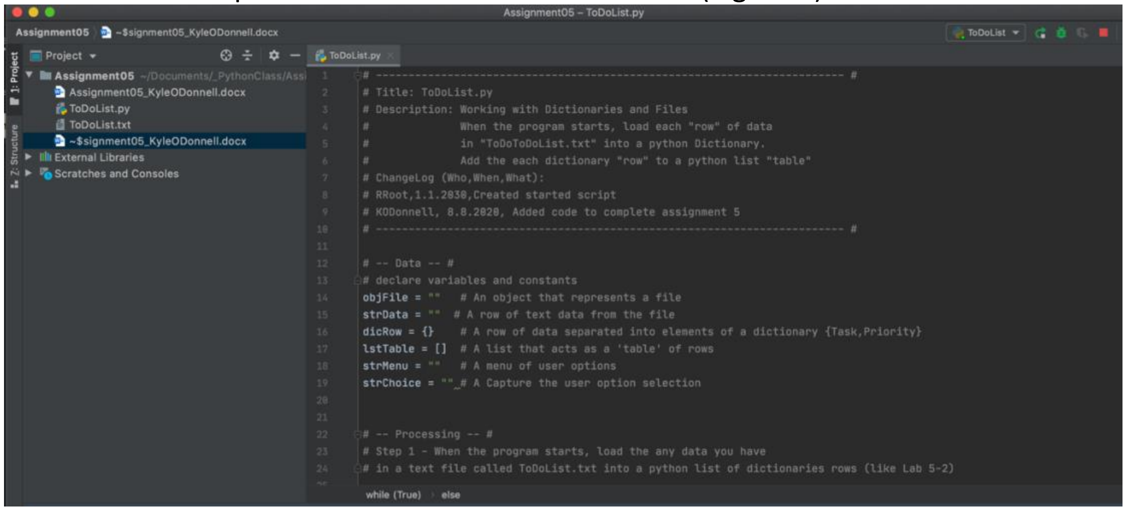
Writing the Script

Template

The script for ‘ToDoList.py’ was adapted from a template provided by Professor Randal Root. The template included metadata and separate sections for “Data,” “Processing,” and “Presentation,” consistent with the design principle of “Separation of Concerns” (SoC). Additionally, the template contained some baseline code and an outline for the branching ‘if’ statements. Although the instructions noted that it can be challenging to work from someone else’s code, I found the template was helpful for mapping out logic and applying SoC concepts.

Pycharm

Similar to previous assignments, this program was written in PyCharm. To get started, I created a new project in the Assignment05 directory called ‘ToDoList.py,’ pasted the provided template into the file and updated the metadata in the header (Figure 1).


Assignment05 – ToDoList.py
Assignment05
~$signment05_KyleODonnell.docx
ToDoList
Project
ToDoList.py
Assignment05 ~/Documents/_PythonClass/Assi
Assignment05_KyleODonnell.docx
ToDoList.py
ToDoList.txt
~$signment05_KyleODonnell.docx
External Libraries
Scratches and Consoles
1: Project
Z: Structure
while (True) › else

```
# ------------------------------------------------------------------------ #
# Title: ToDoList.py
# Description: Working with Dictionaries and Files
#              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
# KODonnell, 8.8.2020, Added code to complete assignment 5
# ------------------------------------------------------------------------ #

# -- Data -- #
# declare variables and constants
objFile = ""   # An object that represents a file
strData = ""  # A row of text data from the file
dicRow = {}    # A row of data separated into elements of a dictionary {Task,Priority}
lstTable = []  # A list that acts as a 'table' of rows
strMenu = ""   # A menu of user options
strChoice = ""_# A Capture the user option selection


# -- Processing -- #
# Step 1 - When the program starts, load the any data you have
# in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)
```

Figure 1. Creating ToDoList.py in PyCharm using the provided template

Data

The first section of code declares the variables and constants that will be used throughout the program. This portion was provided with the template for this assignment.

```
objFile = ""   # An object that represents a file
strData = ""  # A row of text data from the file
dicRow = {}    # A row of data separated into elements of a dictionary {Task,Priority}
lstTable = []  # A list that acts as a 'table' of rows
strMenu = ""    # A menu of user options
strChoice = "" # A Capture the user option selection
```

Listing 1

Not all of these objects need to be declared at the beginning of the script for the program to work properly. However, their inclusion here is consistent with the SoC principle that underlies formatting conventions in Python (and other languages).

Processing

This section loads data from the file to an object within the script.

```
try:
    objFile = open("ToDoList.txt", "r")
    for row in objFile:
        strData = row.split(",")
        dicRow = {"Task":strData[0], "Priority": strData[1].strip()}
        lstTable.append(dicRow)
    objFile.close()
except:
    pass
```

Listing 2

Note that I employ the Try-Except method of error handling to support instances in which the file does not already exist in the same directory as the program. If the file *does* exist, the ‘open’ function opens the file “ToDoList.txt” in ‘read’ mode and assigns it to objFile. A ‘for’ loop converts the file into a program-friendly table by iterating over each ‘row’ and executing the following steps:

1. Each ‘row’ is broken into two elements using ‘split’ to separate values based on commas found in the file. These elements are assigned to strData as a list.
2. The elements of strData are assigned to the dictionary dicRow (the dictionary class is indicated by the use of braces). The first element is assigned to the “Task” key, and the second is assigned to the “Priority” value using list indexing. The ‘strip’ function removes any carriage returns or extra spacing at the end of each line.
3. dicRow is appended to lstTable, generating a list of dictionaries that effectively serves as a table in this program.

Once the ‘for’ loop is complete, the objFile closes. The resulting data in lstTable will be used to complete each menu option available to users. The use of nested dictionaries (as opposed to lists) makes the structure similar to a conventional database in which values can be referenced by their column/row placements.

If ‘ToDoList.txt’ is *not* present in the same directory as the program, the program will skip to the ‘Except’ statement and ‘pass’ to the next section of the program without error.

Presentation

The “Presentation” is the most complex portion of the script and has several subsections covered in detail below. While there is not a strict delineation of “Processing” and “Presentation” here, future versions of the program may provide greater separation of these aspects by using custom functions.

Greeting, ‘While’ Loop, Menu Selection

The beginning of the presentation greets users and elicits their menu option.

```
print("""\n\tWelcome to ToDoList!!
Select an option from the menu to get started!""")
while (True):
   print("""
   Menu of Options
   1) Show current data
   2) Add a new item
   3) Remove an existing item
   4) Save Data to File
   5) Exit Program
   """)
   strChoice = str(input("Which option would you like to perform? [1 to 5] "))
```

Listing 3

The code can be broken into four parts that are all relatively straightforward:

1. Greeting – prints a simple string to greet users when they initiate the program
2. Infinite ‘while’ loop – ensures users stay within program and are continuously prompted with the menu options until they choose to exit
3. Options Menu – prints a string of text that lets users know each option
4. Option Input – elicits input of options 1-5 from users

Show Current List

If users enter “1,” the first conditional statement displays current data in their ‘to do’ list.

```
if (strChoice.strip() == '1'):
    if lstTable == []:
        print("Your To Do list is currently empty.")
    else:
        print("\n\t<<<Current To Do List>>>\n")
        count = 0
        for row in lstTable:
            count +=1
            print(str(count)+". "+ row["Task"].capitalize() + " | " + row["Priority"].upper())
    continue
```

Listing 4

Within the conditional expression, the 'strip()' function removes any spacing that may have been entered with the input. All of the successive branches of the 'while' loop use the 'strip()' method as well.

Within the 'if' statement, I created nested conditional statements. If lstTable has no data, the program will return a message that their 'to do' list is currently empty. This message will display if there was no pre-existing ToDoList.txt file in their directory, or if the file happened to have no data when it was processed.

If lstTable has any data, a 'for' loop is used to print the key and value of each dictionary row within the table. I used a few different methods to format presentation for users, including:

1. The 'capitalize()' method capitalizes the first letter of each task
2. The 'upper()' method capitalizes each letter of the priority
3. The task and priority are separated by " | " using concatenation
4. The object "count" and addition assignment are used to number each row

Add Tasks to List

Selecting option "2" allows users to add new tasks to their list.

```
elif (strChoice.strip() == '2'):
    lstTask = str(input("Please enter a new To Do task:\n "))
    lstPriority = str(input("Please enter the priority for {} (High, Medium or Low):\n ".format(lstTask)))
    dicRow = {"Task": lstTask, "Priority": lstPriority}
    lstTable.append(dicRow)
    print("{} saved to your list!".format(lstTask).capitalize())
    continue
```

Listing 5

Using 'input' statements, the user will be prompted to enter a new task and its priority. The input objects are saved as a dictionary to dicRow, which is appended to lstTable (similar to the process that loaded each row of data from the file to the lstTable earlier). A print statement then confirms the new item has been added to the list.

Remove Data

Users can remove data from their list in by entering “3.”

```
    elif (strChoice.strip() == '3'):
        strItem = str(input("Which item would you like to remove?\n "))
        count =0
        for row in lstTable:
            if row["Task"].lower() == strItem.lower():
                lstTable.remove(row)
                count +=1
        if count > 0: print("{} removed from list.".format(strItem))
        else: print("Item not found")
        continue
```

Listing 6

An ‘input’ statement elicits the name of a task they would like to remove. A ‘for’ loop then iterates over each row of the listTable to identify dictionary rows with matching tasks. The ‘remove’ function deletes any rows with matches from the table, and the ‘lower’ function handles any variation in casing.

I have also used a ‘count’ object with addition assignment to count each deleted row. If the count is greater than “0,” the program confirms the item has been deleted. Otherwise, the program prints a statement that the item was not found.

Save Data To File

Option “4” allows users to save the data to a file.

```
    elif (strChoice.strip() == '4'):
        objFile = open("ToDoList.txt", "w")
        for row in lstTable:
            objFile.write(row["Task"].capitalize() + ", " + row["Priority"].upper()+"\n")
        print("Data Saved to File!")
        objFile.close()
        continue
```

Listing 7

If users want to save their data to a file, the program re-opens the ToDoList.txt file in ‘write’ mode and assigns it to the object objFile. It then uses a ‘for’ loop to write each row of lstTable as a string to objFile. The Task and Priority are separated by a comma using concatenation and followed by a carriage return. After each row is written to the file, the object is closed.

Note that the current data in lstTable will overwrite the existing data in the file during this step. Because that data was loaded into the lstTable during the “Processing” stage, anything that was not deliberately modified while the program was used will be preserved and re-written to the file. Additionally, if there was no pre-existing ‘ToDoList.txt’ in the same directory as the program, this step will create a new file and save data to it.

Close Program

If users select the last option, “5”, they exit the program.

```
elif (strChoice.strip() == '5'):
    print("Goodbye!\n")
    break  # Exit the program
```

Listing 8

The program prints a “Goodbye” message and breaks out of the ‘while’ loop to exit.

Other Inputs

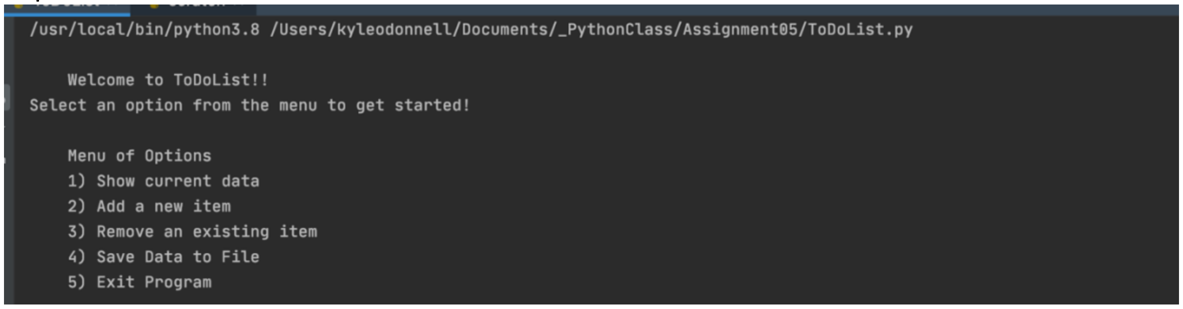
The final ‘else’ clause handles inputs that are not part of the options menu. If users enter something other than 1, 2, 3, 4 or 5, the program prints a statement and returns to the menu.

```
else:
    print("Please select an option from the menu.")
    continue
```

Listing 9

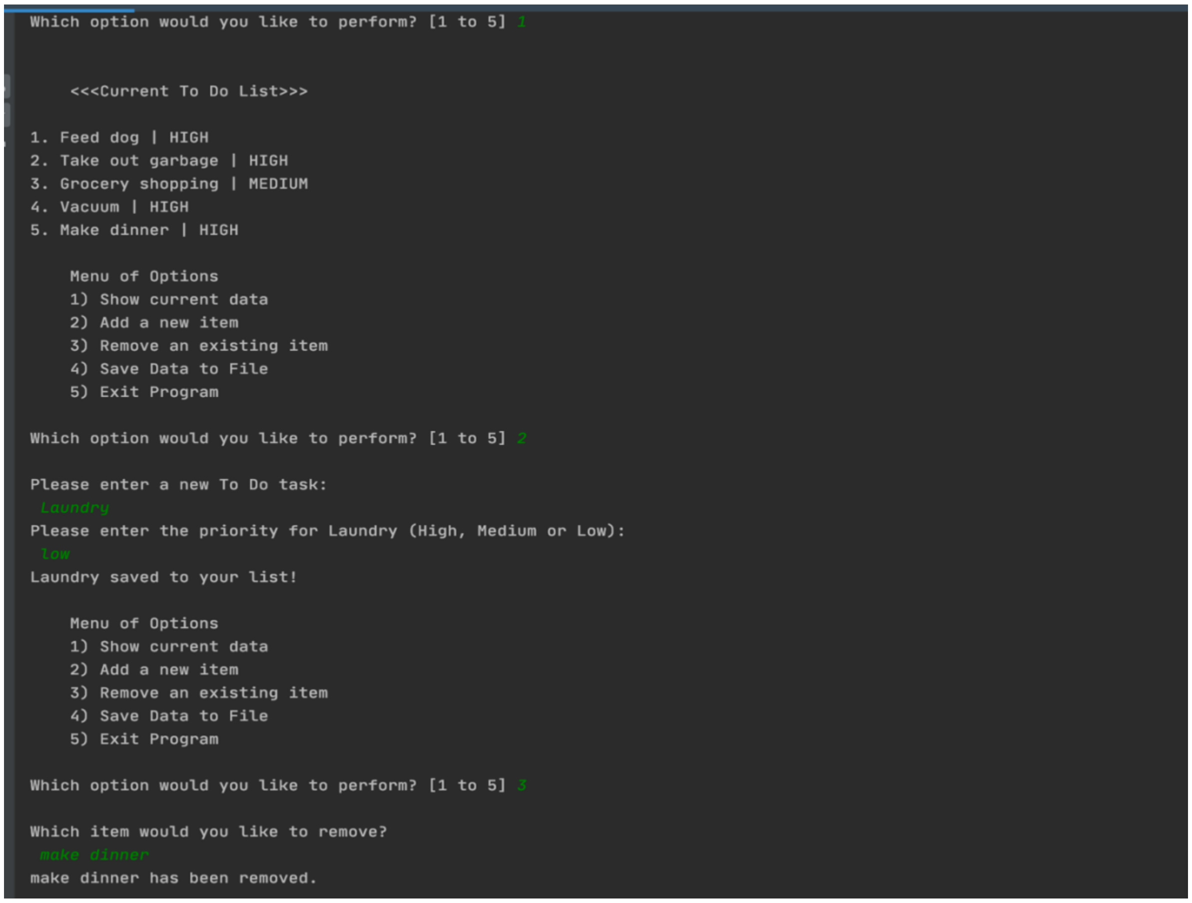
Testing the Script in PyCharm

After completing the script, I tested it in PyCharm. Figures 2-4 show each option working as expected.



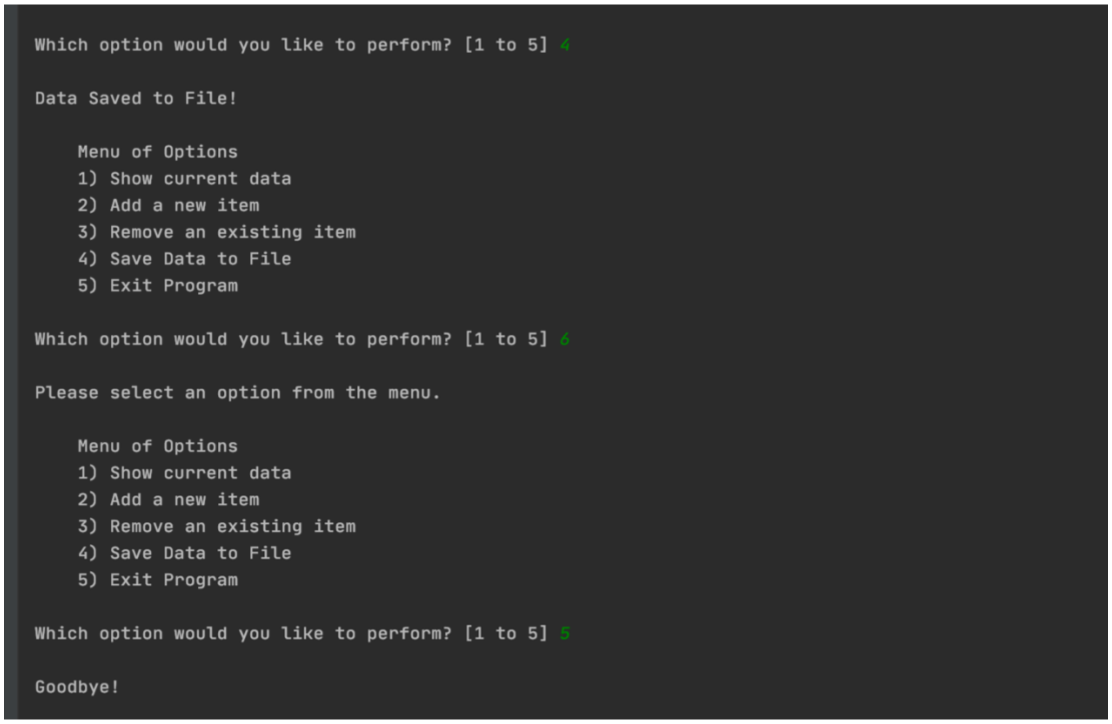
```
/usr/local/bin/python3.8 /Users/kyleodonnell/Documents/_PythonClass/Assignment05/ToDoList.py

    Welcome to ToDoList!!
Select an option from the menu to get started!

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program
```

Figure 2. Launching the script in Pycharm



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


    <<<Current To Do List>>>

1. Feed dog | HIGH
2. Take out garbage | HIGH
3. Grocery shopping | MEDIUM
4. Vacuum | HIGH
5. Make dinner | HIGH

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

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

Please enter a new To Do task:
 Laundry
Please enter the priority for Laundry (High, Medium or Low):
 low
Laundry saved to your list!

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

Which option would you like to perform? [1 to 5] 3

Which item would you like to remove?
 make dinner
make dinner has been removed.
```

Figure 3. Viewing list, adding to list and removing from list

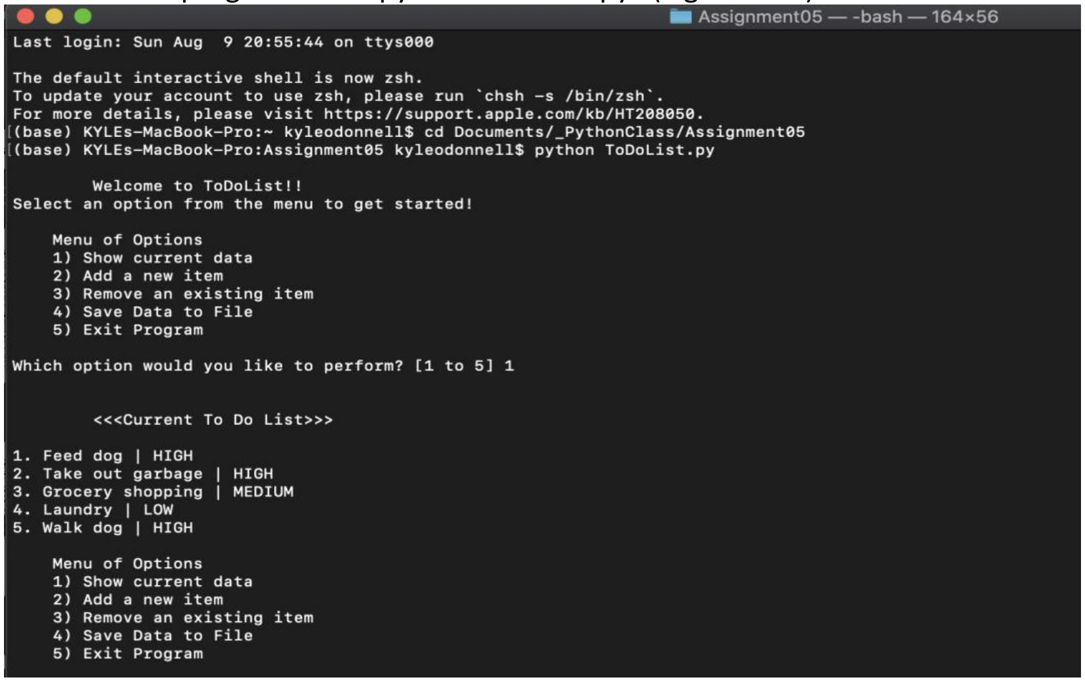


```
Which option would you like to perform? [1 to 5] 4

Data Saved to File!

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

Which option would you like to perform? [1 to 5] 6

Please select an option from the menu.

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

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

Goodbye!
```

Figure 4. Saving data to file, invalid inputs and exiting

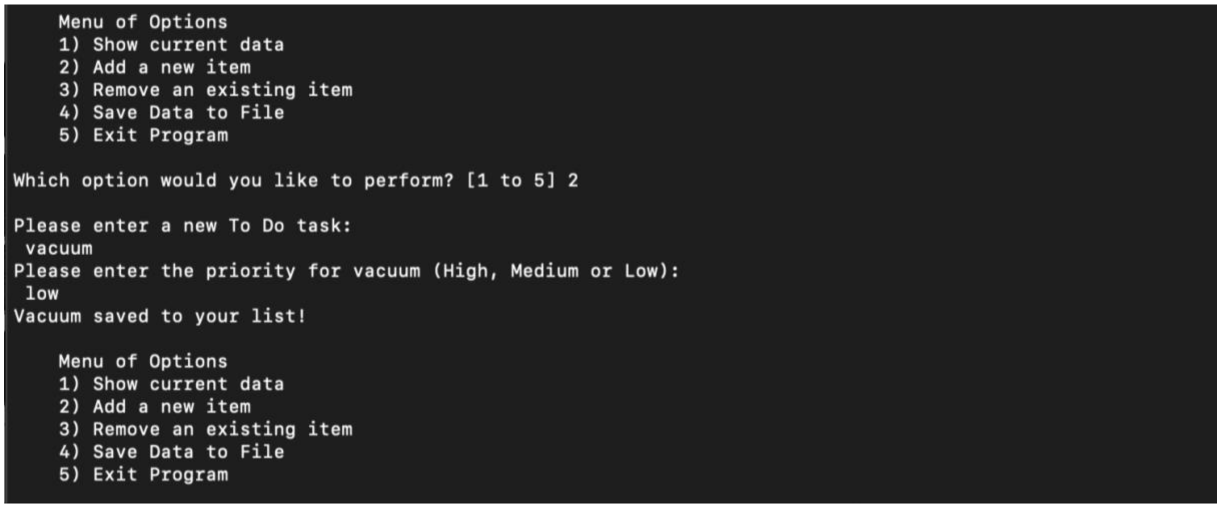
Running the Script in Terminal

I next ran the program as a console application in Terminal. To do this, I navigated to the relevant directory with the command “cd Documents/_PythonClass/Assignment05” and then launched the program with “python ToDoList.py” (Figures 5-7).



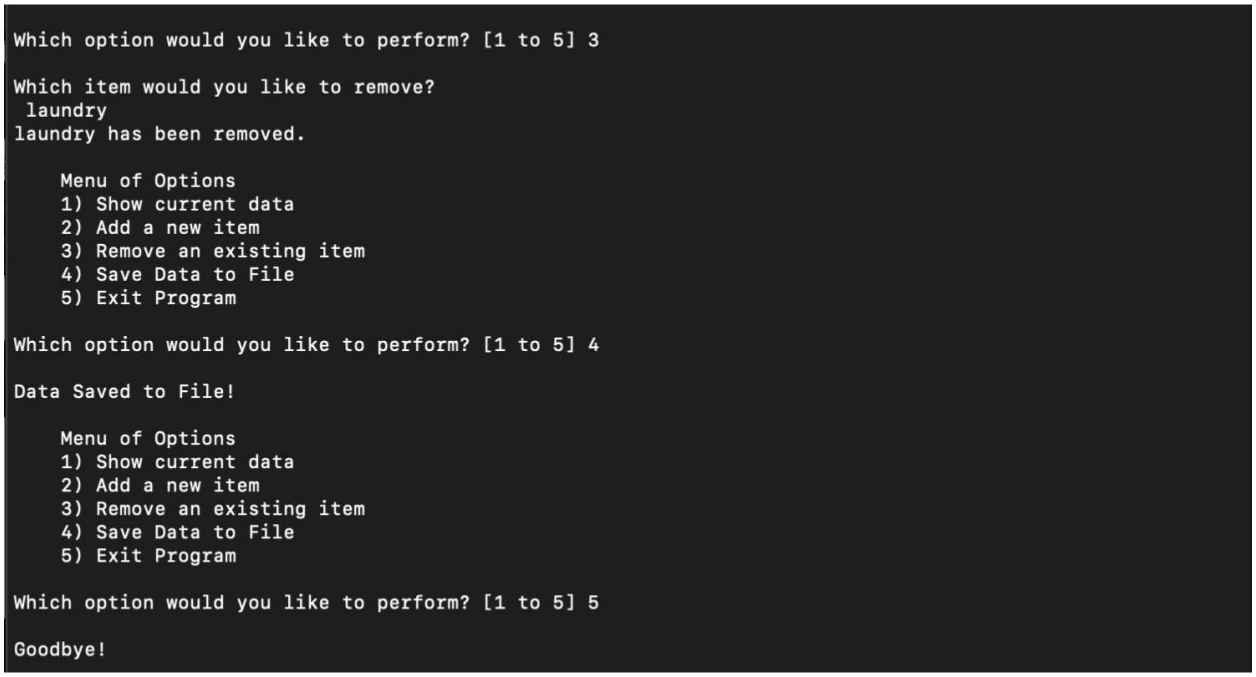
```
Assignment05 — -bash — 164×56
Last login: Sun Aug  9 20:55:44 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) KYLEs-MacBook-Pro:~ kyleodonnell$ cd Documents/_PythonClass/Assignment05
(base) KYLEs-MacBook-Pro:Assignment05 kyleodonnell$ python ToDoList.py

        Welcome to ToDoList!!
Select an option from the menu to get started!

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

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


        <<<Current To Do List>>>

1. Feed dog | HIGH
2. Take out garbage | HIGH
3. Grocery shopping | MEDIUM
4. Laundry | LOW
5. Walk dog | HIGH

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program
```

Figure 5. Launching ToDoList.py program in Terminal and viewing list



```
    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

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

Please enter a new To Do task:
 vacuum
Please enter the priority for vacuum (High, Medium or Low):
 low
Vacuum saved to your list!

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program
```

Figure 6. Adding items to list

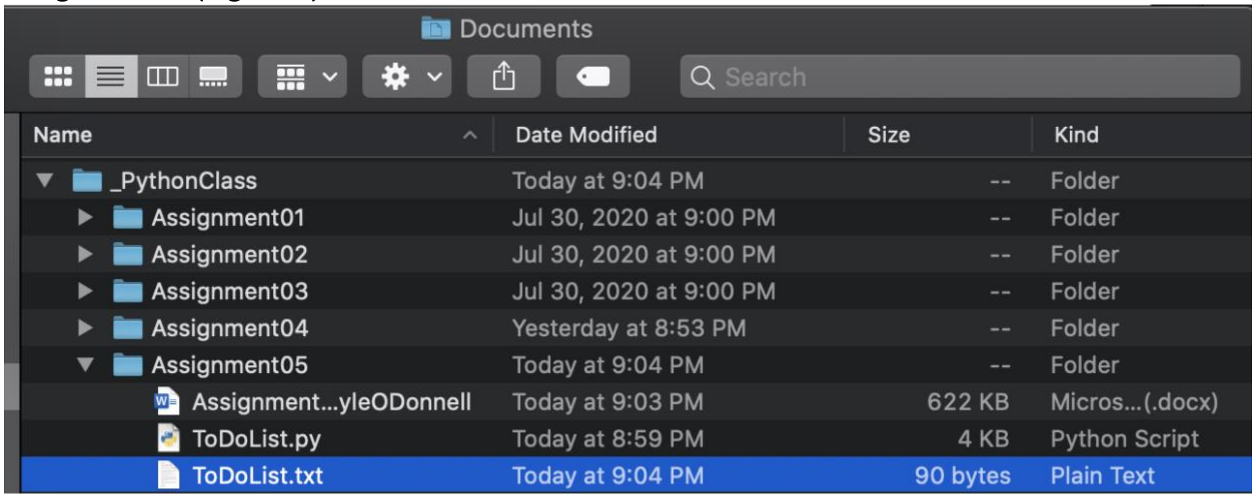


```
Which option would you like to perform? [1 to 5] 3

Which item would you like to remove?
 laundry
laundry has been removed.

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

Which option would you like to perform? [1 to 5] 4

Data Saved to File!

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Remove an existing item
    4) Save Data to File
    5) Exit Program

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

Goodbye!
```

Figure 7. Removing items from list, saving to file and closing

Text File

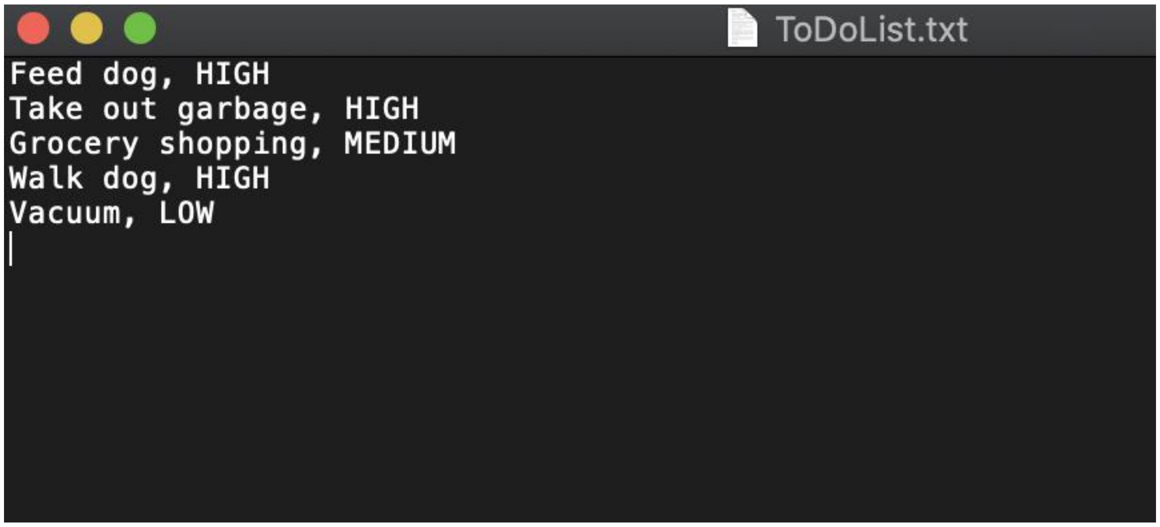
To confirm that step 4 really saved the data to the file, I checked the file 'ToDoList.txt' in the Assignment05 (Figure 8).



Name	Date Modified	Size	Kind
_PythonClass	Today at 9:04 PM	--	Folder
Assignment01	Jul 30, 2020 at 9:00 PM	--	Folder
Assignment02	Jul 30, 2020 at 9:00 PM	--	Folder
Assignment03	Jul 30, 2020 at 9:00 PM	--	Folder
Assignment04	Yesterday at 8:53 PM	--	Folder
Assignment05	Today at 9:04 PM	--	Folder
Assignment...yleODonnell	Today at 9:03 PM	622 KB	Micros...(.docx)
ToDoList.py	Today at 8:59 PM	4 KB	Python Script
ToDoList.txt	Today at 9:04 PM	90 bytes	Plain Text

Figure 8. ToDoList.txt in Assignment05 directory

The data I had added, removed and saved while running the program in Terminal were all saved in the Assignment05 directory (Figure 9).



```
Feed dog, HIGH
Take out garbage, HIGH
Grocery shopping, MEDIUM
Walk dog, HIGH
Vacuum, LOW
```

Figure 9. Data saved to ToDoList.txt from ToDoList.py

Some potential enhancements for future versions of the program include:

1. Recognizing more robust menu inputs, such as "one" or "option 1."
2. Allowing users to exit the program at any point by typing "exit."
3. Sorting the list by priority.
4. Providing a drop down menu or buttons for selecting the priority.

Summary

This paper describes the process of creating the ‘ToDoList.py’ program in Python. The program allows users to open, edit and save data to a ‘to do’ list. The program was based on a template created for this assignment. The final product was tested in PyCharm and Terminal, and the data was confirmed to have successfully been saved to the file ‘ToDoList.txt.’