Corbin Bridge

November 15, 2022

IT FDN 110 A Au 22

Assignment05

https://github.com/cbridge-uw/IntroToProg-Python

The To Do List Script

Introduction

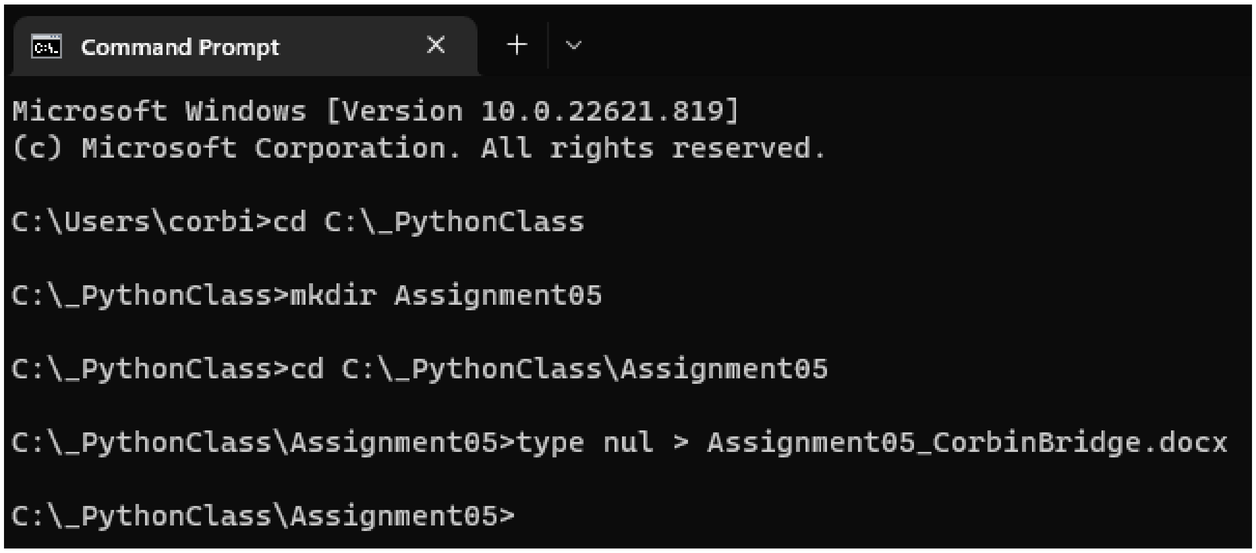
The To Do List script will utilize a Python dictionary to allow users to build a to do list complete with tasks and priorities. This knowledge document will outline the steps taken to create the script. In the final section of the document, I will introduce GitHub and demonstrate adding files to a repository.

The To Do List Script

For this script, the user must be able to add a task to a to do list and set a priority. While working with their list, the user should be able to view the current data, add a new task, remove a task, and save the data to a text.

Initial Set Up

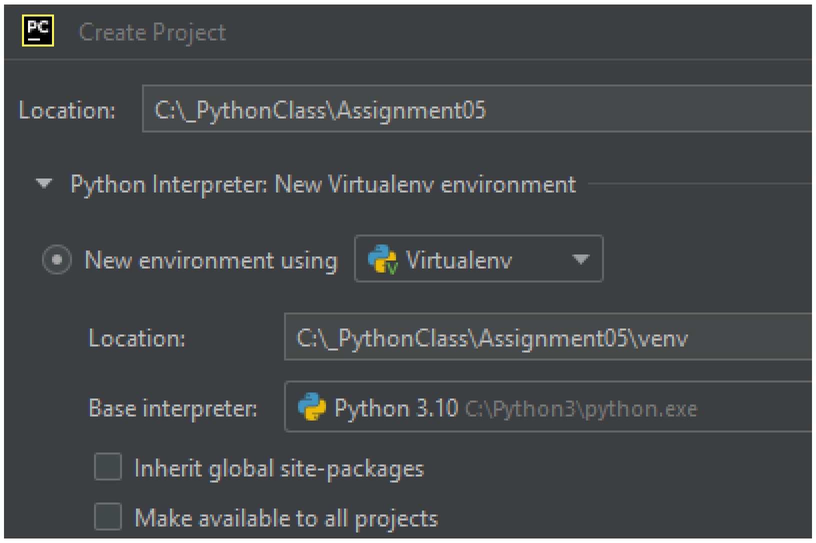
The initial set up follows the pattern of prior assignments. I created the Assignment05 sub-folder in the _PythonClass folder and created a Word document (Figure 1).



```
Microsoft Windows [Version 10.0.22621.819]
(c) Microsoft Corporation. All rights reserved.

C:\Users\corbi>cd C:\_PythonClass

C:\_PythonClass>mkdir Assignment05

C:\_PythonClass>cd C:\_PythonClass\Assignment05

C:\_PythonClass\Assignment05>type nul > Assignment05_CorbinBridge.docx

C:\_PythonClass\Assignment05>
```

Figure 1: Creating the Assignment05 folder and Word document.

Next, I created a new project in PyCharm (Figure 2).


Create Project

Location: C:_PythonClass\Assignment05

Python Interpreter: New Virtualenv environment

New environment using: Virtualenv

Location: C:_PythonClass\Assignment05\venv

Base interpreter: Python 3.10 C:\Python3\python.exe

☐ Inherit global site-packages

☐ Make available to all projects

Figure 2: Creating a new PyCharm project.

I found the “Assignment05_Starter.py” file that was included in the Assignment05 zip file and added it to the project (Figure 3).

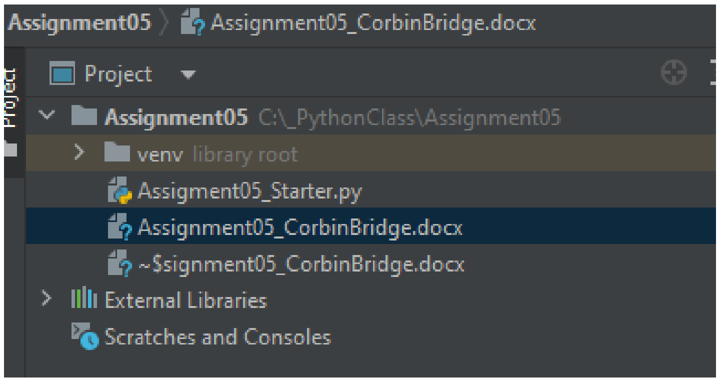
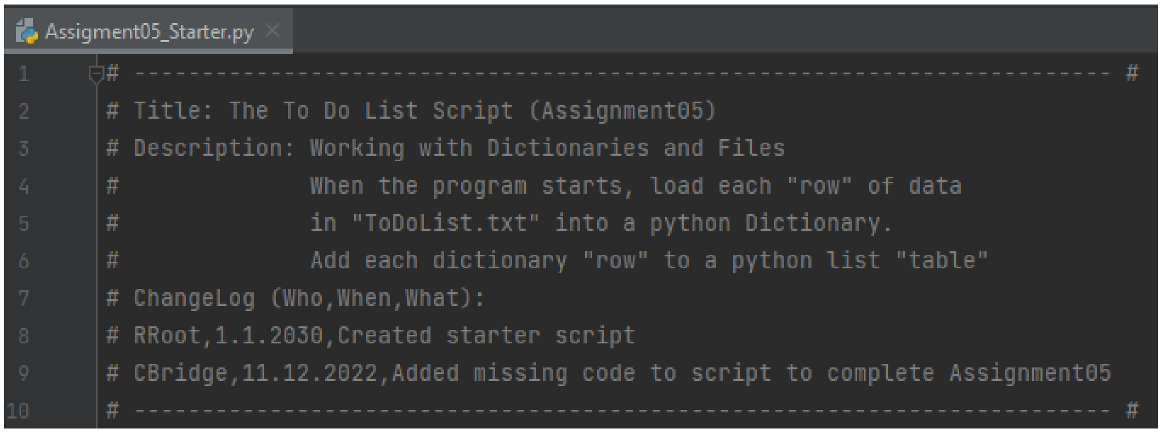

```
Assignment05 〉 Assignment05_CorbinBridge.docx
Project
Assignment05 C:\_PythonClass\Assignment05
    venv library root
    Assigment05_Starter.py
    Assignment05_CorbinBridge.docx
    ~$signment05_CorbinBridge.docx
External Libraries
Scratches and Consoles
```

Figure 3: Adding the “Assignment05_Starter.py” file.

The final stage of the initial setup was to edit the script header section (Figure 4).

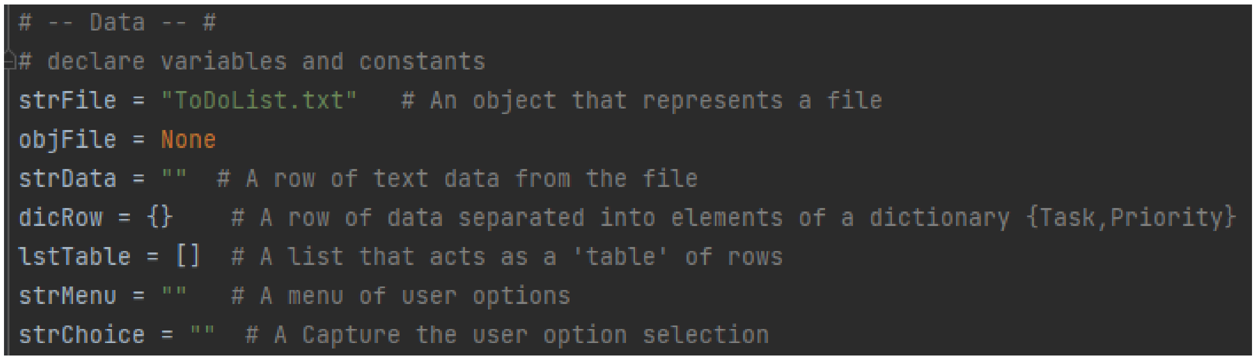

Assigment05_Starter.py

```
# ------------------------------------------------------------------------ #
# Title: The To Do List Script (Assignment05)
# Description: Working with Dictionaries and Files
#              When the program starts, load each "row" of data
#              in "ToDoList.txt" into a python Dictionary.
#              Add each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created starter script
# CBridge,11.12.2022,Added missing code to script to complete Assignment05
# ------------------------------------------------------------------------ #
```

Figure 4: Modifying the script header.

Reviewing the Data Section

Unlike prior assignments, the To Do list script comes with a starter file. Before writing any code, I reviewed the variables in the data section (Figure 5). I felt it was important to understand the variables that had been declared, their data types, and what, if any, values they were initialized with.



```
# -- Data -- #
# declare variables and constants
strFile = "ToDoList.txt"   # An object that represents a file
objFile = None
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
```

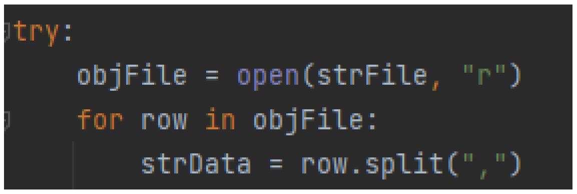
Figure 5: Variables provided in the data section.

I added the strFile variable and set it to represent the ToDoList.txt file. objFile was then set to None and will be used to handle the file in the script.

Updating the Processing Section

The processing section begins with a prompt and a TODO item, but no code.

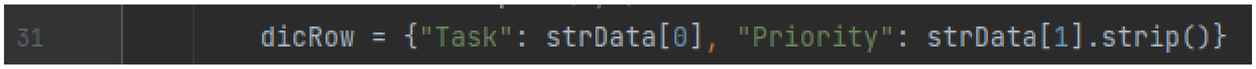
First, I opened ToDoList.txt in read mode. I then split each row of text saved in the file using the split() method. Since we are working with a comma separated value text file, the comma was declared as the delimiter (Figure 6).



```
try:
    objFile = open(strFile, "r")
    for row in objFile:
        strData = row.split(",")
```

Figure 6: Opening ToDoList.txt and reading into dictionary rows.

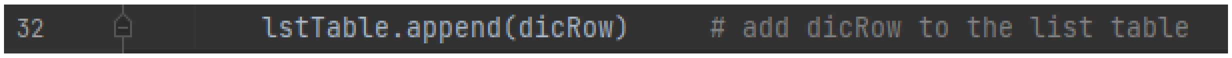
Next, I read each row from strData into a Python list using the dicRow variable. As dictionaries use key: value pairs, I declared “Task” and “Priority” as the keys with their respective values as position 0 and 1 in strData. The strip() method was also applied to remove any carriage returns present in the text file. (Figure 7).



```
            dicRow = {"Task": strData[0], "Priority": strData[1].strip()}
```

Figure 7: Reading data into a Python dictionary row.

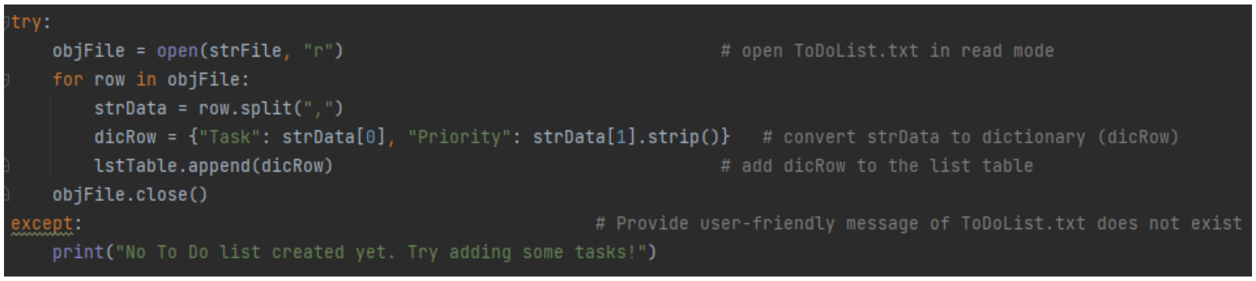
The final step was to add each of the dictionary rows, contained in the dicRow variable, to the list table represented using the lstTable variable (Figure 8). This was accomplished using the append() method.



```
            lstTable.append(dicRow)     # add dicRow to the list table
```

Figure 8: Adding dictionary rows to lstTable.

I included try/except error handling to avoid returning a system generated message if ToDoList.txt was not yet created (Figure 9). The system error messages are not user friendly, so I added a user-friendly message. Using try/except will allow the code to continue running and avoid any confusion for the end user if a ToDoList.txt file does not exist when they start the program.



```
try:
    objFile = open(strFile, "r")                                  # open ToDoList.txt in read mode
    for row in objFile:
        strData = row.split(",")
        dicRow = {"Task": strData[0], "Priority": strData[1].strip()}   # convert strData to dictionary (dicRow)
        lstTable.append(dicRow)                                   # add dicRow to the list table
    objFile.close()
except:                                                     # Provide user-friendly message of ToDoList.txt does not exist
    print("No To Do list created yet. Try adding some tasks!")
```

Figure 9: Try Except error handling.

If a ToDoList.txt file exists, the user will not receive any message from the processing section. The menu of options will be the first prompt to greet the user.

Collecting User Input

The input/output section of the script presents the user with a menu of options. This menu was included in the starter file and is similar to the menu created in the HomeInventory.py file in Assignment04.

The following sections detail the steps taken for each menu option.

Using strip() and Continue

Throughout the script the reader will notice the use of the strip() method and continue statement.

Strip() is used to remove any leading or lagging spaces added by the user. Python is picky in that “1 “ (notice the lagging white space) is not equal to “1” (no spaces). If a user accidently presses the space bar it would result in an error. Strip() removes these spaces and helps to provide a better user experience.

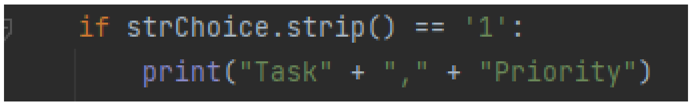
Continue is a statement that tells Python to return to the top of the while loop. It is not necessary, but Professor Root mentioned that it does help other programmers understand my code.

As strip() and continue are included in multiple areas of the scrip this section serves to address their use. I will not discuss them at each occurrence.

Show Current Data

If the user selects menu option 1, they will be shown the current list of data in the to do list.

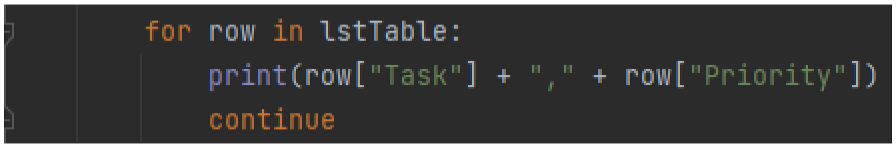
The first task is to check the user’s input. If the option is 1 then a simple header of “Task” and “Priority” will print to the screen (Figure 10).



```
    if strChoice.strip() == '1':
        print("Task" + "," + "Priority")
```

Figure 10: Checking user selection and printing the list header.

Next, I use a for loop to print out the tasks and priority currently contained in lstTable (Figure 11).

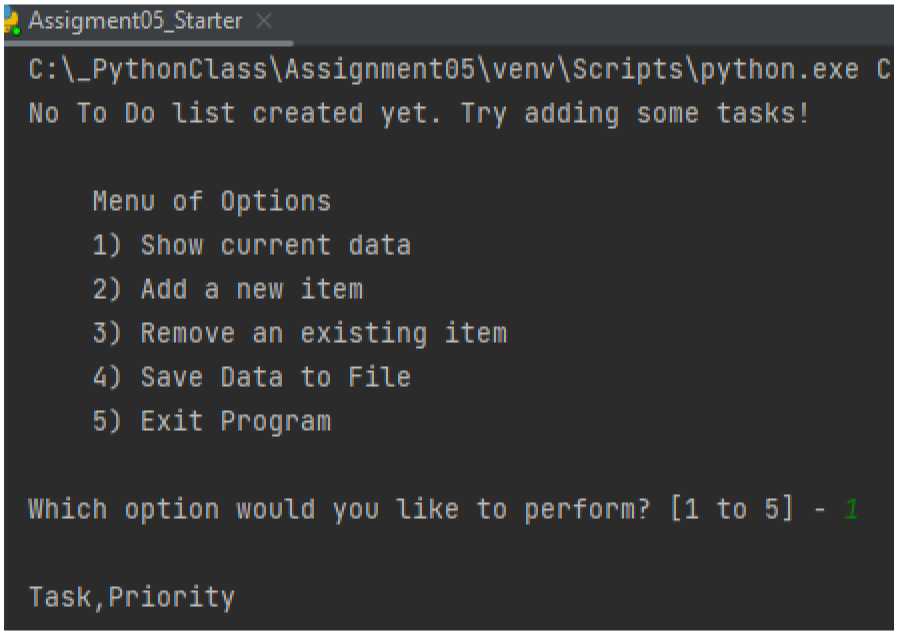


```
        for row in lstTable:
            print(row["Task"] + "," + row["Priority"])
            continue
```

Figure 11: Printing tasks and their priority.

As the table contains a list of dictionary rows I pass in the key, “Task” and “Priority” in Figure 11, and Python returns the corresponding value. This is akin to accessing a list index with a numeric subscript, but with a dictionary a string representing the key is used. Personally, “Task” is much easier to understand than [0].

A quick test confirmed that the code was running as expected thus far (Figure 12).



```
C:\_PythonClass\Assignment05\venv\Scripts\python.exe C
No To Do list created yet. Try adding some tasks!

    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

Task,Priority
```

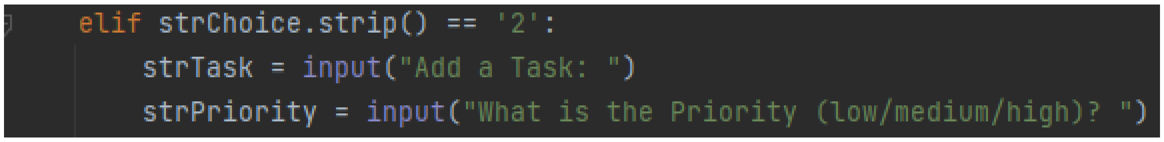
Figure 12: Testing the show current data functionality.

The next step is to add a new item to the list.

Add a New Item

In this stage it will be necessary for a user to add a new task and priority to the to do list.

First, I check if the user’s selection is 2. If true, then I request a task and priority from the user (Figure 13).

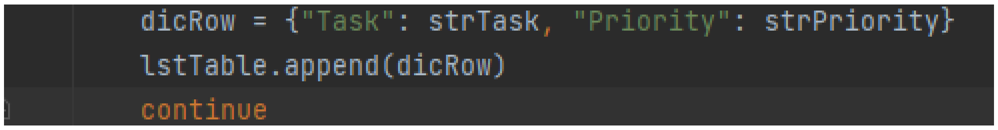


```
    elif strChoice.strip() == '2':
        strTask = input("Add a Task: ")
        strPriority = input("What is the Priority (low/medium/high)? ")
```

Figure 13: Testing user selection and requesting a task and priority from the user.

Since I am declaring two new variables, task and priority, I add them to the data section and initialize both as empty strings.

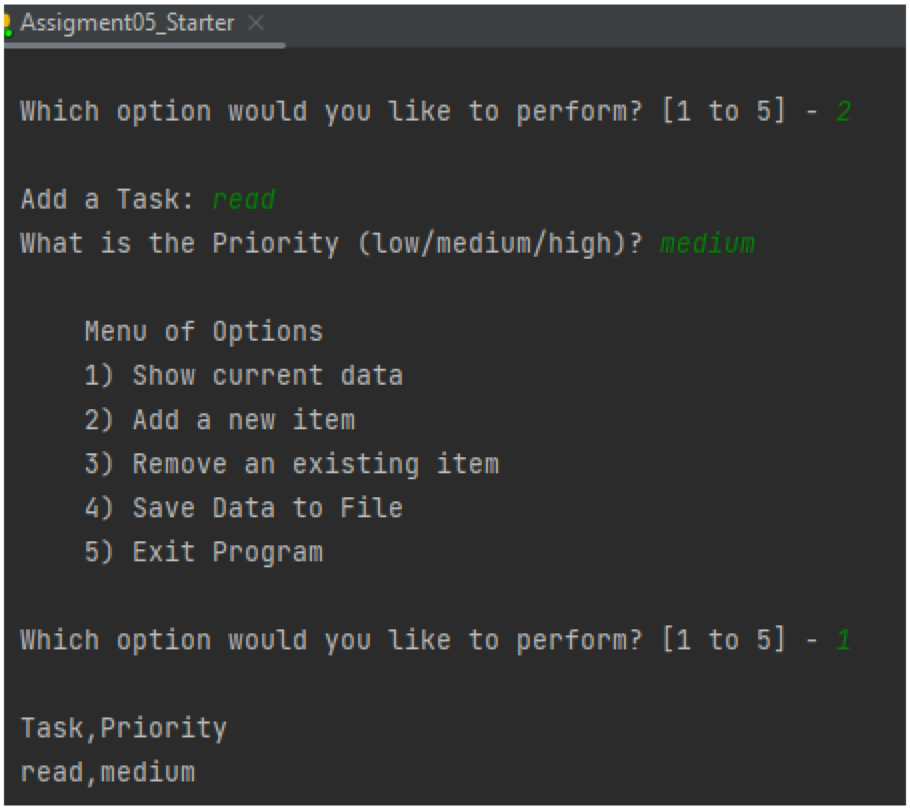
Once the user has supplied the necessary inputs both task and priority are converted to a dictionary row. Finally, the new dictionary row is appended to lstTable (Figure 14).



```
        dicRow = {"Task": strTask, "Priority": strPriority}
        lstTable.append(dicRow)
        continue
```

Figure 14: Create dictionary row with user input and append to the table.

Before moving on I conducted a quick test to ensure the program was running as planned (Figure 15).



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

Add a Task: read
What is the Priority (low/medium/high)? medium

    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

Task,Priority
read,medium
```

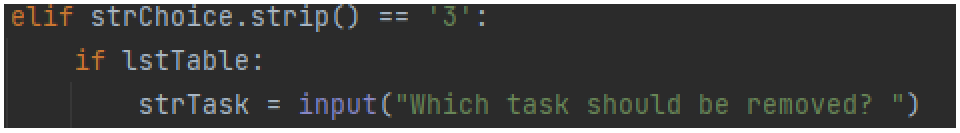
Figure 15: Testing the user entry code.

I was able to enter the new task and priority and then print out the current list.

Remove an Existing Item

Hopefully our user is completing some of their tasks! If so, they will need the ability to remove them from the to do list.

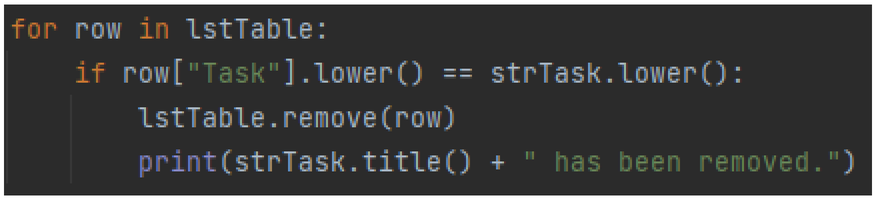
If the user selection is 3, they will be prompted to enter the task they want to remove (Figure 16).



```
elif strChoice.strip() == '3':
    if lstTable:
        strTask = input("Which task should be removed? ")
```

Figure 16: Testing user selection and requesting the task to be removed.

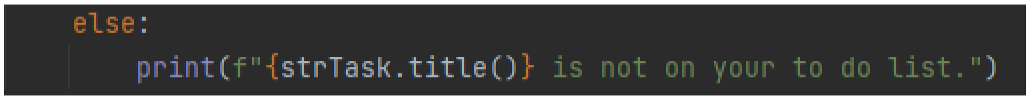
I started with a for loop that would iterate through each item in the list. If the user-provided task was associated with a task in the dictionary list, it would be removed using the remove() method. Finally, a success message would print informing the user that the task was removed (Figure 17).



```
for row in lstTable:
    if row["Task"].lower() == strTask.lower():
        lstTable.remove(row)
        print(strTask.title() + " has been removed.")
```

Figure 17: Finding and deleting a user-defined task from the list.

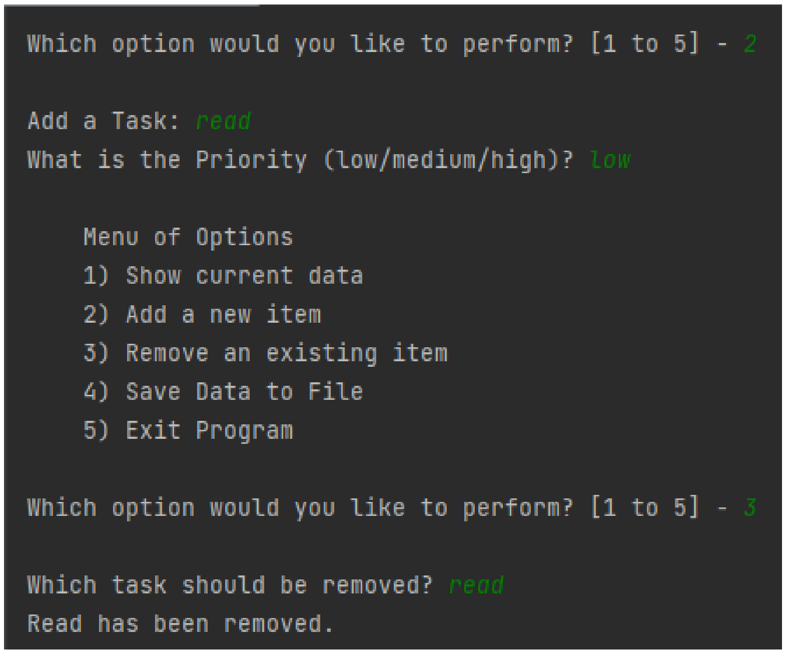
If the user has requested to remove a task that does not exist, they are told that the task is not on the to do list (Figure 18).



```
    else:
        print(f"{strTask.title()} is not on your to do list.")
```

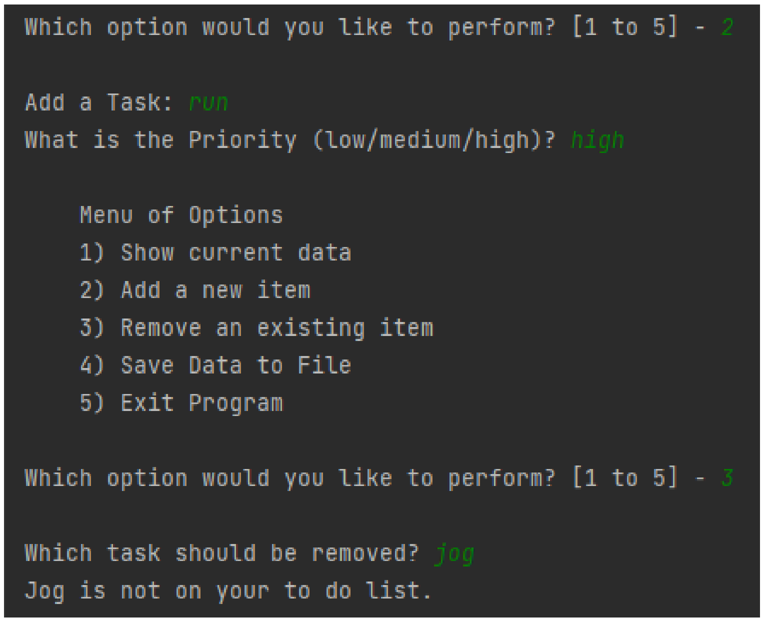
Figure 18: Informing user the task is not on the to do list.

I once again test removing a task (Figure 19) and returning the else message (Figure 20) before proceeding to the next step.



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

Add a Task: read
What is the Priority (low/medium/high)? low

    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 task should be removed? read
Read has been removed.
```

Figure 19: Removing a task.



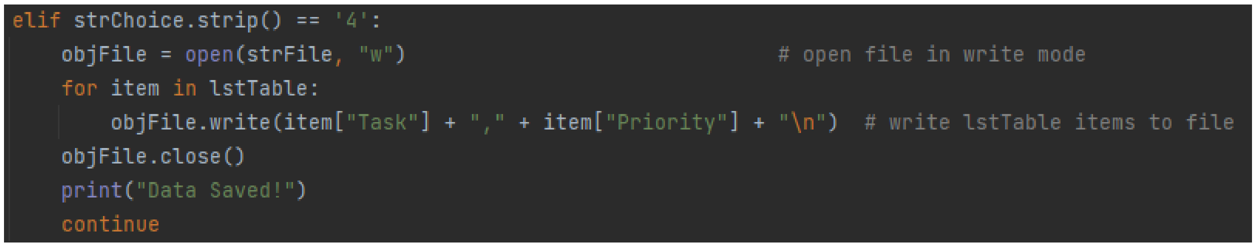
```
Which option would you like to perform? [1 to 5] - 2

Add a Task: run
What is the Priority (low/medium/high)? 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] - 3

Which task should be removed? jog
Jog is not on your to do list.
```

Figure 20: Notice that a task is not in the to do list.

Save Data to File

Once the user has entered and removed some tasks, they need the ability to save the file.

I first opened ToDoList.txt (the strFile variable) in write mode and then created a for loop that would write each task and priority into the text file separated by a comma. The final step was to close the file and confirm to the user that the data had been saved (Figure 21).



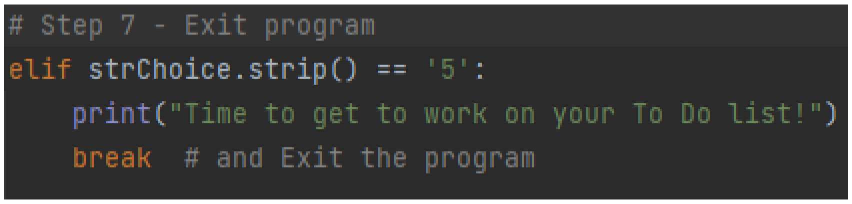
```
elif strChoice.strip() == '4':
    objFile = open(strFile, "w")                            # open file in write mode
    for item in lstTable:
        objFile.write(item["Task"] + "," + item["Priority"] + "\n")  # write lstTable items to file
    objFile.close()
    print("Data Saved!")
    continue
```

Figure 21: Saving data to ToDoList.txt.

Exit Program

The last step for the script is allowing the user to exit the program.

I tested the user selection and, if the result is 5, they are presented with an exit message and the program ends (Figure 22).



```
# Step 7 - Exit program
elif strChoice.strip() == '5':
    print("Time to get to work on your To Do list!")
    break  # and Exit the program
```

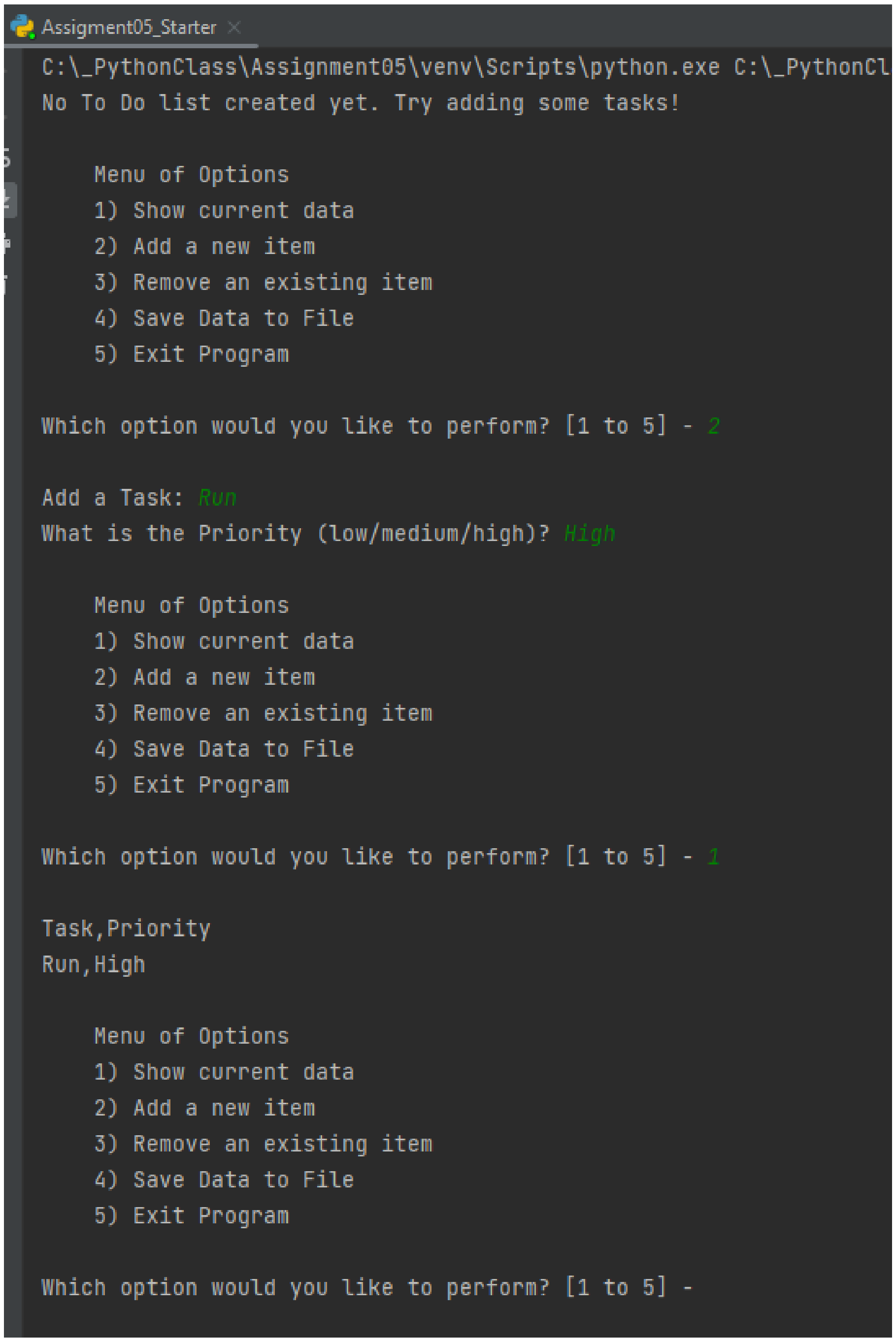
Figure 22: Exiting the program.

Testing the Script

With the script created it was time to test it out!

PyCharm Test

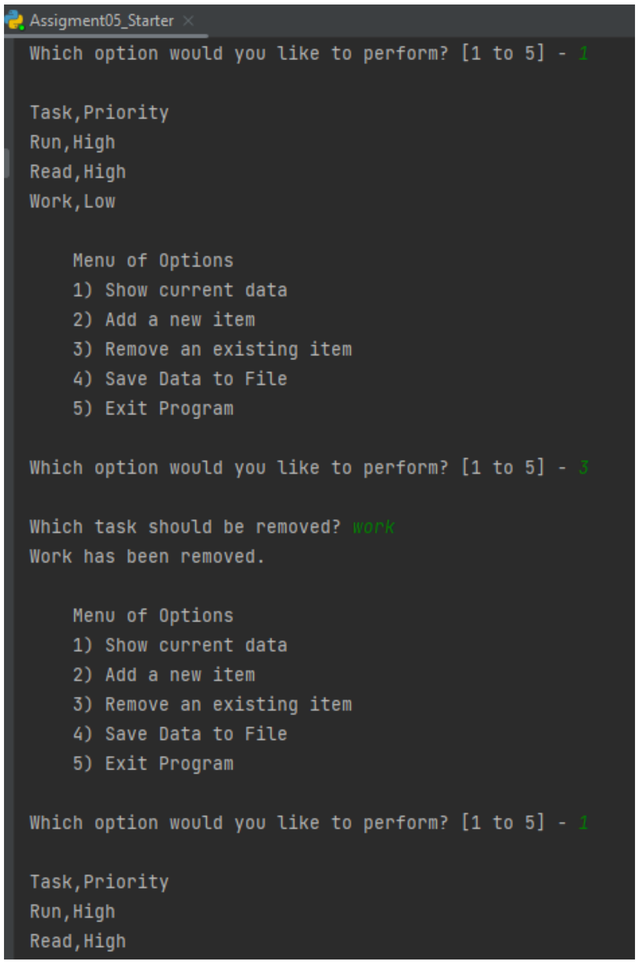
The first test was to run the script in PyCharm and ensure I could add an item with option 2 and then view the current list with option 1. I was also checking to ensure that my friendly message using the except statement worked properly (Figure 23).



```
C:\_PythonClass\Assignment05\venv\Scripts\python.exe C:\_PythonCl
No To Do list created yet. Try adding some tasks!

    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

Add a Task: Run
What is the Priority (low/medium/high)? 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] - 1

Task,Priority
Run,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] -
```

Figure 23: Testing adding, viewing, and except statement.

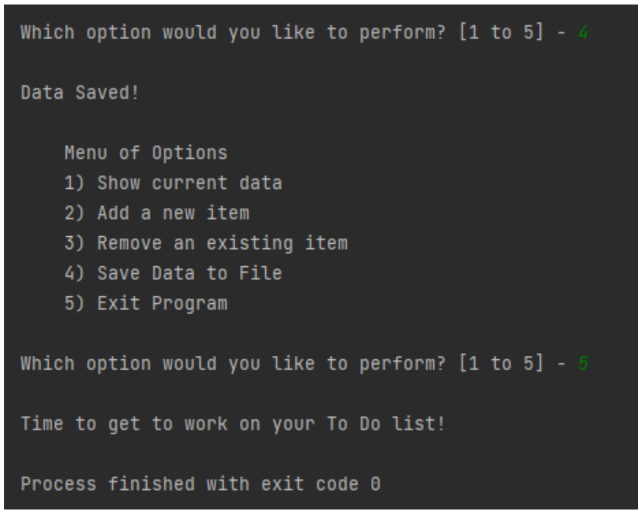
I added a few more items and then tested the option 3, removing an item from the list (Figure 24).



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

Task,Priority
Run,High
Read,High
Work,Low

    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 task should be removed? work
Work 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] - 1

Task,Priority
Run,High
Read,High
```

Figure 24: Testing remove.

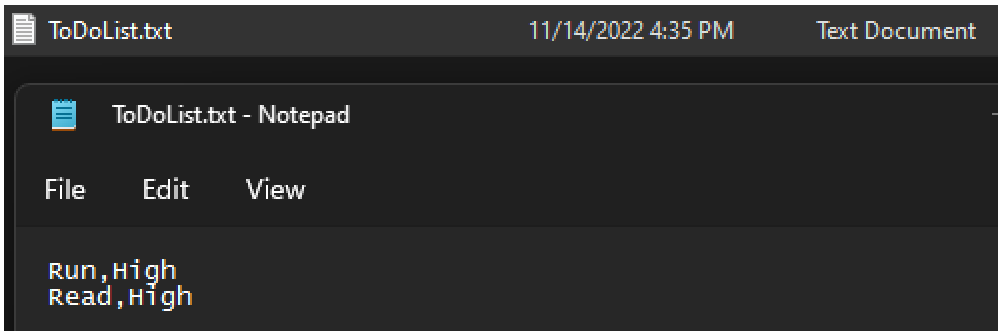
Finally, I saved the file and exited the program (Figure 25).



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

Data Saved!

    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

Time to get to work on your To Do list!

Process finished with exit code 0
```

Figure 24: Save the list and exit the program.

I navigated to the ToDoList.txt file to ensure the data was saved properly (Figure 25).



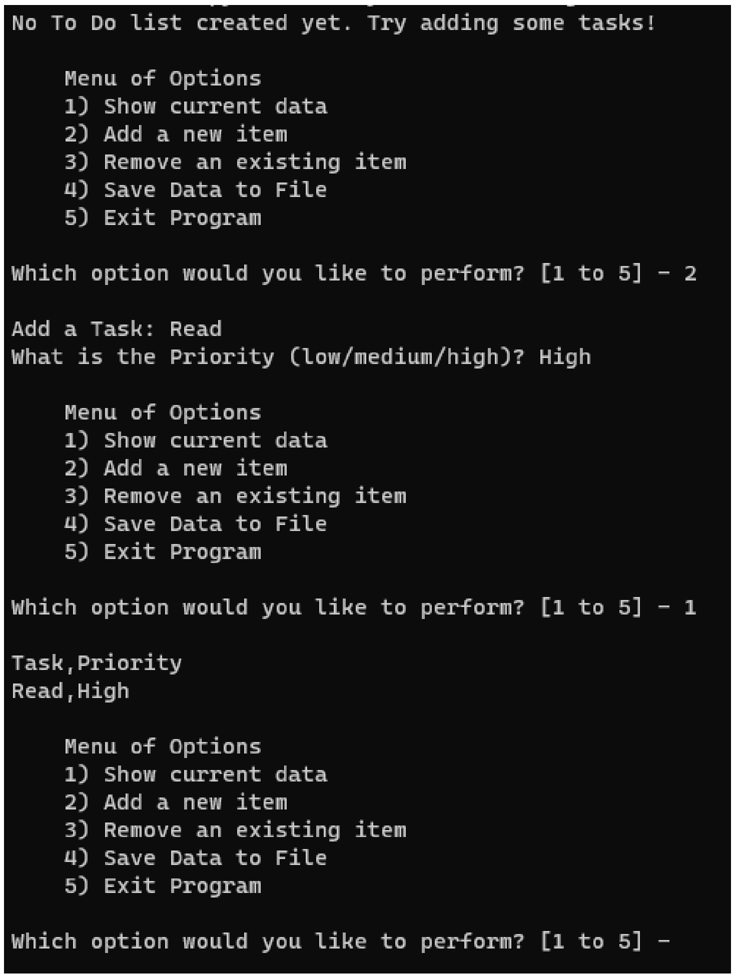
```
Run,High
Read,High
```

Figure 25: Verifying the data was properly saved.

Command Shell Test

With a successful PyCharm test I next opened a command window (Windows + cmd) and passed the file path.

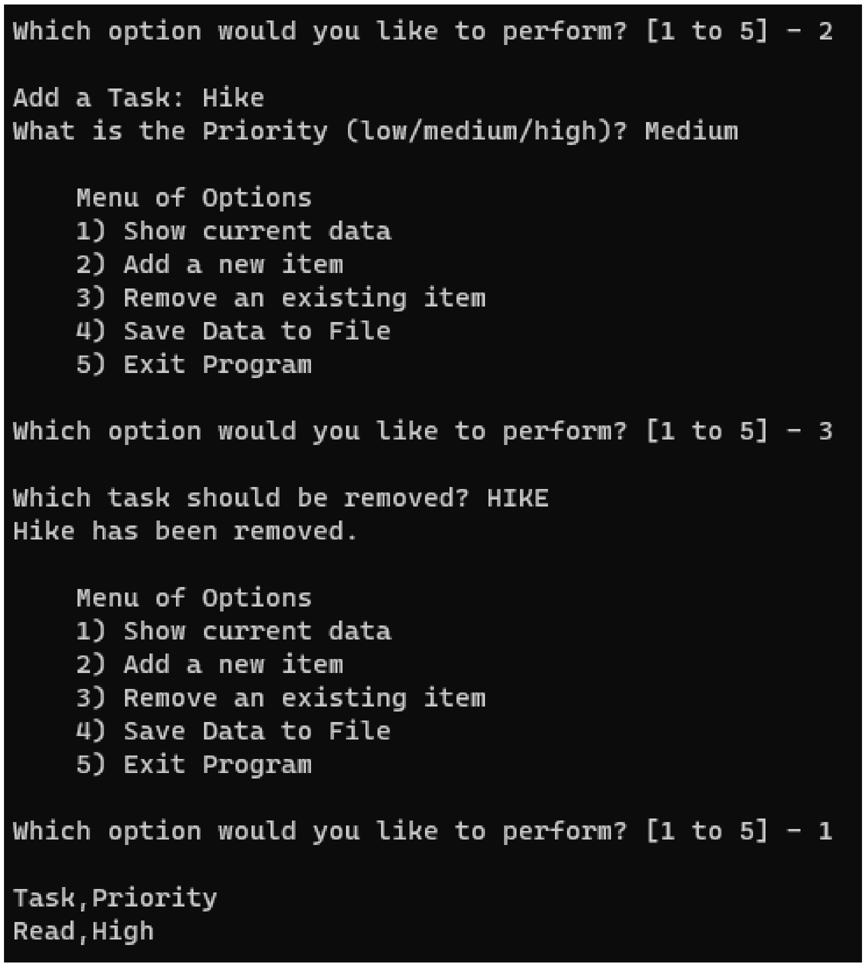
The program started and I tested the ability to add a task and view the current list (Figure 26).



```
No To Do list created yet. Try adding some tasks!

    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

Add a Task: Read
What is the Priority (low/medium/high)? 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] - 1

Task,Priority
Read,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] -
```

Figure 26: Adding a task and viewing the current list.

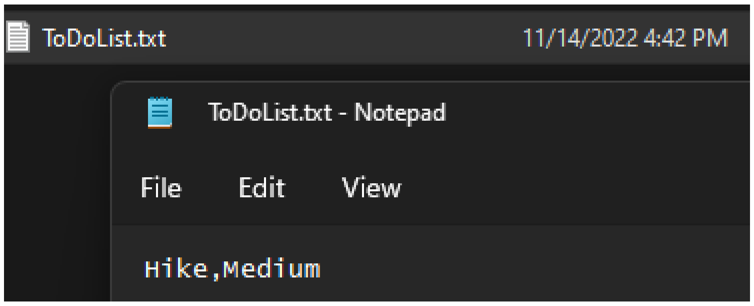
For the next step I added an additional task, removed it, then confirmed that it was removed by checking the current list (Figure 27).



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

Add a Task: Hike
What is the Priority (low/medium/high)? Medium

    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 task should be removed? HIKE
Hike 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] - 1

Task,Priority
Read,High
```

Figure 27: Testing adding and removing a task.

The last step was to save the data and find the file in the proper folder (Figure 28).



```
Hike,Medium
```

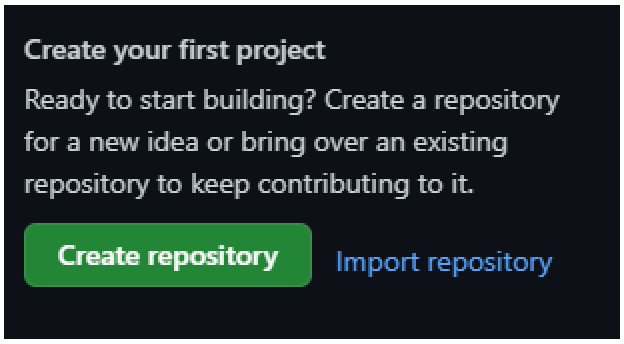
Figure 28: Confirming the file saved.

GitHub

For this assignment we are required to create a GitHub profile and save our project. Creating a GitHub account follows a standard process used when creating most online accounts (enter an email, username, etc.). Once the account was created I needed to create a repository.

Creating a Repository

GitHub provides a useful "Create Repository" button when you login (Figure 29).

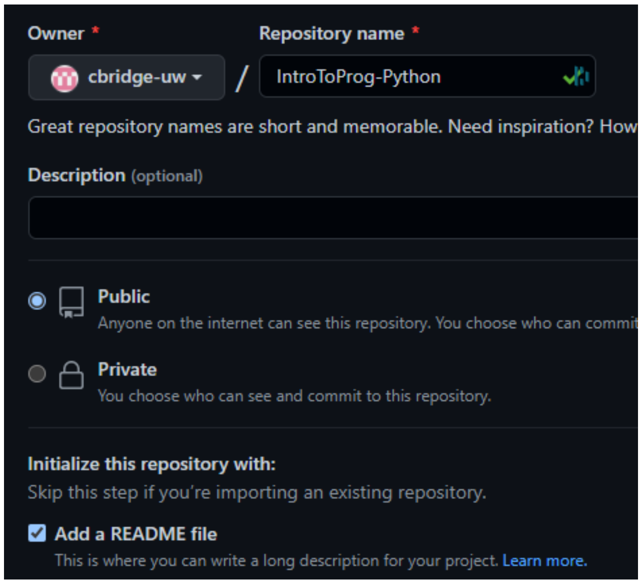


Create your first project

Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

Figure 29: Create repository in GitHub.

Following the link presents the user with an easy setup menu. I entered the repo name, ensured it was set to Public, and initialized with a README file (Figure 30).



Owner * cbridge-uw / Repository name * IntroToProg-Python

Great repository names are short and memorable. Need inspiration? How

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit

Private You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

[✓] Add a README file

This is where you can write a long description for your project. Learn more.

Figure 30: Creating a GitHub repo.

Adding Files to the Repo

To add a file, I selected the dropdown menu and selected Upload files (Figure 31).

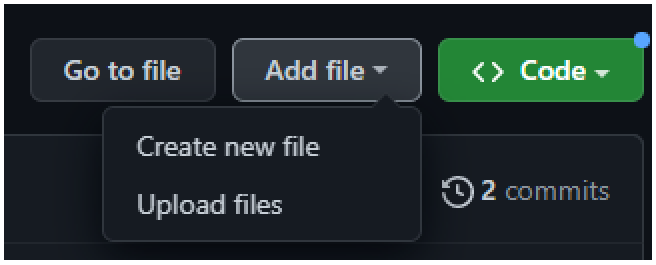


Figure 31: Adding files to a repo.

I selected the ToDoList.py file, added a commit note and then committed the changes (Figure 32).

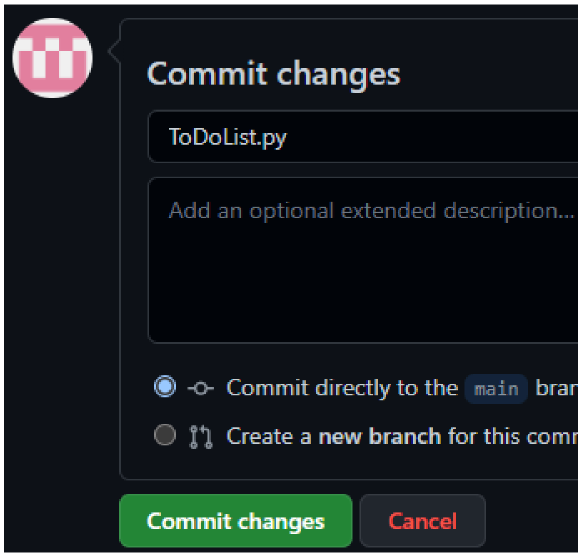

Commit changes
ToDoList.py
Add an optional extended description...
Commit directly to the main bran
Create a new branch for this comn
Commit changes
Cancel

Figure 32: Committing changes to the repo.

Summary

In this assignment I have utilized lists and dictionaries to create an interactive to do list. This was also the first assignment where GitHub was used. While more challenging than prior assignments, it is rewarding to see how skills developed in prior lessons are starting to enable the development of an interactive and handy script.