Craig Muth

11/14/2022

IT FDN 110 A Au 22: Foundations Of Programming: Python

Assignment 05

https://github.com/cjmuth/IntroToProg-Python

TODO List - Read/Write an External File, and the GitHub Repository

Introduction

The goal of this project is to create a TODO list program that will read data from an existing text file, allow the user to view the existing data, make changes, and save the data back to the text file.

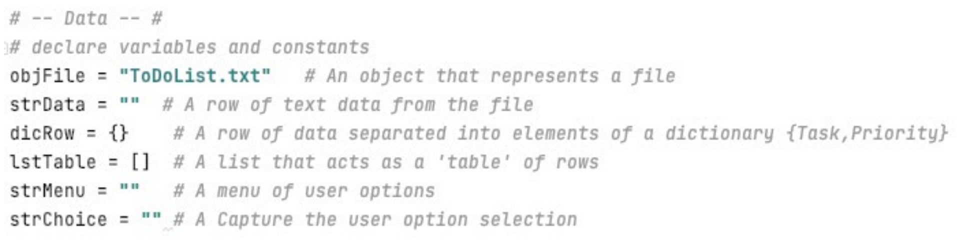
The program will be organized according to a “Separation of Concerns” pattern to help group sections of code according to the types of operations they perform.

Finally the project files will be uploaded to a GitHub repository to make them available to other users, but still maintain control and history of revisions.

Designing the program

For this project, a starter file was provided that lays out the basic structure and some of beginning code for portions of the program. The program will be divided into three sections for data, processing and input/output.

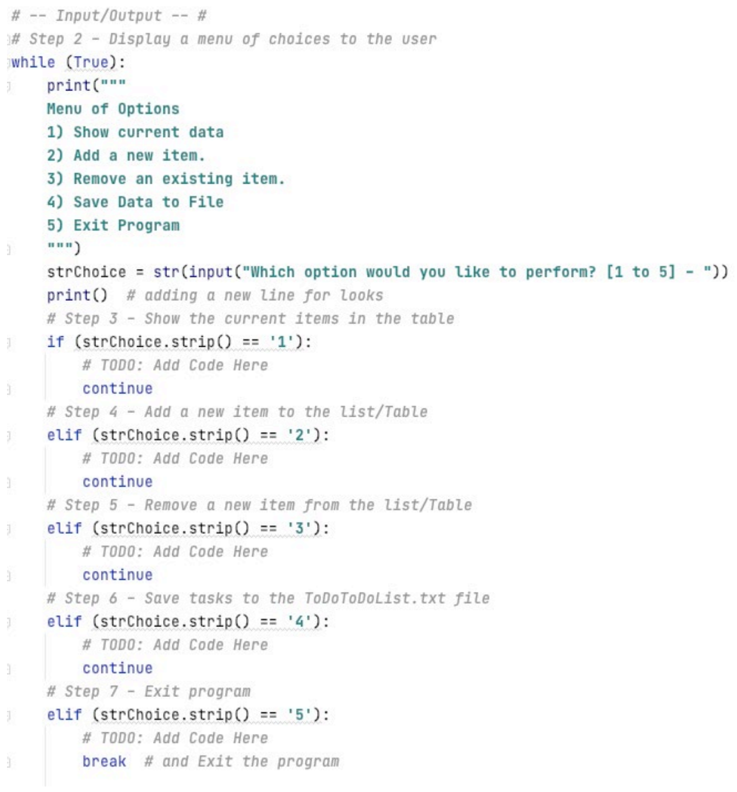
A set of variable definitions was provided for the data section, as seen here.



```
# -- Data -- #
# declare variables and constants
objFile = "ToDoList.txt"   # 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
```

Figure 1: Given variable definitions

Structure for a menu to inform the users of options, collect their input, and controlling the flow of the program was also included in the starter file.



```
# -- Input/Output -- #
# Step 2 - Display a menu of choices to the user
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] - "))
    print()  # adding a new line for looks
    # Step 3 - Show the current items in the table
    if (strChoice.strip() == '1'):
        # TODO: Add Code Here
        continue
    # Step 4 - Add a new item to the list/Table
    elif (strChoice.strip() == '2'):
        # TODO: Add Code Here
        continue
    # Step 5 - Remove a new item from the list/Table
    elif (strChoice.strip() == '3'):
        # TODO: Add Code Here
        continue
    # Step 6 - Save tasks to the ToDoToDoList.txt file
    elif (strChoice.strip() == '4'):
        # TODO: Add Code Here
        continue
    # Step 7 - Exit program
    elif (strChoice.strip() == '5'):
        # TODO: Add Code Here
        break  # and Exit the program
```

Figure 2: Given code for menu structure

The processing section of the starter provided only comments, and no executable code - so it is not included here.

The logic for the program in this project can be shown as:

- Data section
    - Define variables and constants (given with starter)

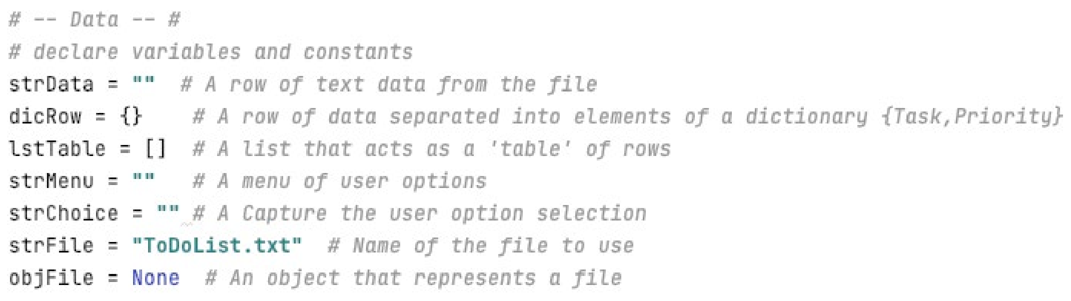
- Processing section
    - Read the file
        - Open the file in read mode
        - Loop through rows in data file
            - Read data row from file as list
            - Save list data to dictionary
            - Add dictionary to table list
        - When end of rows, close file

- Input/Output section
    - Display the menu (given with starter)
    - Get user option

- If option == ‘1’
    - Loop through dictionaries in list
        - Print values to screen
- If option == ‘2’
    - Get user inputs as dictionary
        - Add new dictionary to table list
- If option == ‘3’
    - Get user input for item to remove
    - Remove item from list table list
- If option == ‘4’
    - Open the file in write mode
    - Loop through dictionaries in table list
        - Extract values, concatenate as string, write to text file
    - Close text file
- If option == ‘5’
    - Exit program

Data

Looking at the original code, we can see a variable `objFile` was created to represent the file, but was given a string value. So the first action will be replacing the `objFile` with a generic object rather than the original string value, and creating a separate string variable for the filename.



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

Figure 3: Updated variable definitions

No additional changes to this section are needed at this time.

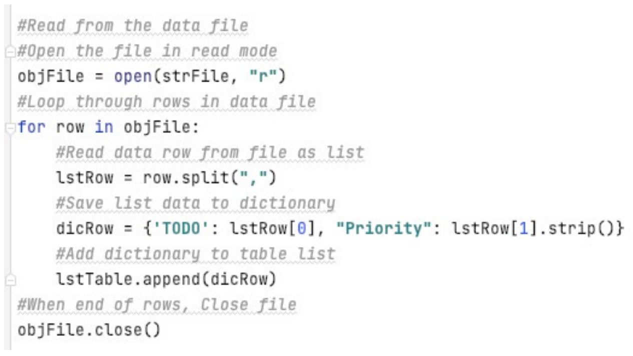
Processing

In this section the only thing we will do is read existing data from a text file.

The file is opened in read mode and assigned to `objFile` - which is then used by a for `loop` to read the data into memory. Since the data was originally written to the file as strings, the `split()` method is used to divide them into lists. Because the original data use a comma to separate the discrete values we have given this as the delimiter on which to separate the data (*Python String split() Method,* (n.d.). Retrieved November 13, 2022, from https://www.w3schools.com/python/ref_string_split.asp).

With the data now in a list, we can use index values to extract the individual elements and assign them to a dictionary with the appropriate key values that can be used to address them later. (*Python - Dictionary* (n.d.). Retrieved November 13, 2022, from https://www.tutorialspoint.com/python/python_dictionary.htm) (External site).

The individual dictionaries are then added to a list - allowing all the data to be passed from one section of code to another with a single reference.



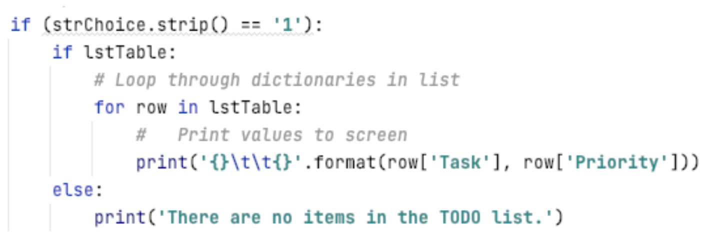
```
#Read from the data file
#Open the file in read mode
objFile = open(strFile, "r")
#Loop through rows in data file
for row in objFile:
    #Read data row from file as list
    lstRow = row.split(",")
    #Save list data to dictionary
    dicRow = {'TODO': lstRow[0], "Priority": lstRow[1].strip()}
    #Add dictionary to table list
    lstTable.append(dicRow)
#When end of rows, Close file
objFile.close()
```

Figure 4: Code for reading the data

Input/Output

After the existing data has been read into memory - the program will then execute the code containing the menu, which also contains most of the processing code in addition to that required for managing the user interactions.

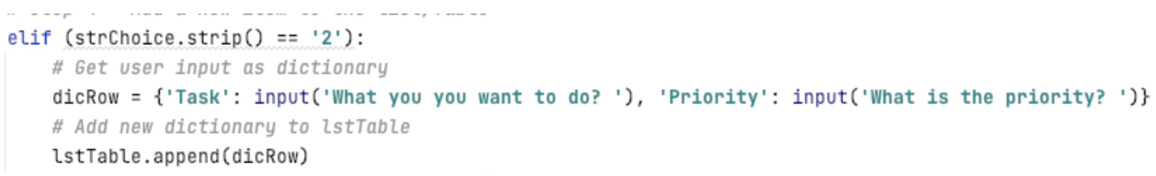
To display the data we will use a `for` loop to iterate through `lstTable` and print out the contents of each dictionary within it. As the program will include an option to remove items from the list, it's possible that a user could remove all the items - which would then cause the program to throw an exception if they chose to display the existing data. Therefore, the code includes an `if…else` statement to first check if there is any data to display, if not it will provide a message to alert the user instead of crashing.



```
if (strChoice.strip() == '1'):
    if lstTable:
        # Loop through dictionaries in list
        for row in lstTable:
            #   Print values to screen
            print('{}\t\t{}'.format(row['Task'], row['Priority']))
    else:
        print('There are no items in the TODO list.')
```

Figure 5: Code to display the data

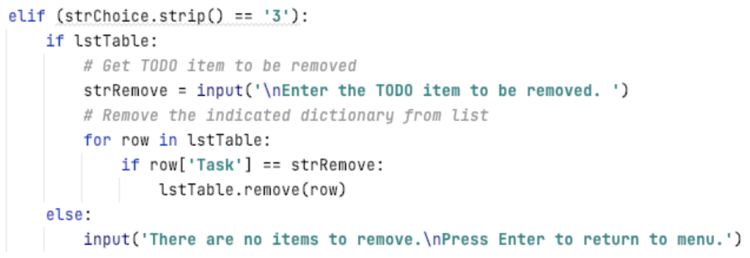
The next option, when selected, prompts the user for information and feeds that input directly into a dictionary, then adds it to the list. It could have used a list to collect the data, similar to the code reading the file - but this eliminates the need for a conversion.



```
elif (strChoice.strip() == '2'):
    # Get user input as dictionary
    dicRow = {'Task': input('What you you want to do? '), 'Priority': input('What is the priority? ')}
    # Add new dictionary to lstTable
    lstTable.append(dicRow)
```

Figure 6: Code to get new entries

The remove option - like the display option - will only work if there is data in `lstTable`. Therefore it contains a comparable `if…else` statement to verify the data exists and notify the user if not.



```
elif (strChoice.strip() == '3'):
    if lstTable:
        # Get TODO item to be removed
        strRemove = input('\nEnter the TODO item to be removed. ')
        # Remove the indicated dictionary from list
        for row in lstTable:
            if row['Task'] == strRemove:
                lstTable.remove(row)
    else:
        input('There are no items to remove.\nPress Enter to return to menu.')
```

Figure 7: Code to remove entries

With the data in a list, we could address the value to remove using the index - if we knew what it was. Since the dataset doesn't have provisions to map the dictionaries to their indices, we need to use a different approach.

Instead, after the user gives their input, the program loops through the list checking the value of `Task` against the user input. When a match is found, the list item is then passed to the `remove()` function (devanshigupta1304 (2022, Nov 9). *How to remove and item from the list in Python?* https://www.geeksforgeeks.org/how-to-remove-an-item-from-the-list-in-python/) (External site.), and eliminated from the list in memory.

When the save option is selected we will open the file in write mode, which will cause the existing file to be overwritten. Since all the data in the file was read into memory, nothing should be lost.

If the file was opened in append mode, the user could still add new items, but the items they removed would not go away and be reloaded the next time the program is run - and new copies of the existing items would be added to the list each time the user saves.

With the text file open, a `for` loop causes the data to be written to the file line by line - then closes the file when the values from the last dictionary are extracted, converted to a string, and saved to the text file.

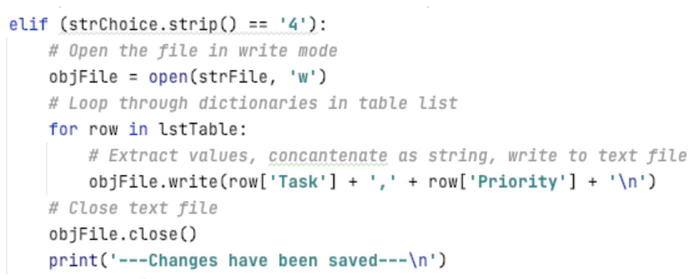

```
elif (strChoice.strip() == '4'):
    # Open the file in write mode
    objFile = open(strFile, 'w')
    # Loop through dictionaries in table list
    for row in lstTable:
        # Extract values, concantenate as string, write to text file
        objFile.write(row['Task'] + ',' + row['Priority'] + '\n')
    # Close text file
    objFile.close()
    print('---Changes have been saved---\n')
```

Figure 8: Code to save data

Since there is already a separate option to save the data, we'll leave the final selection as is and let the program exit and close the console without an additional message this time.

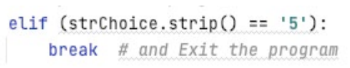
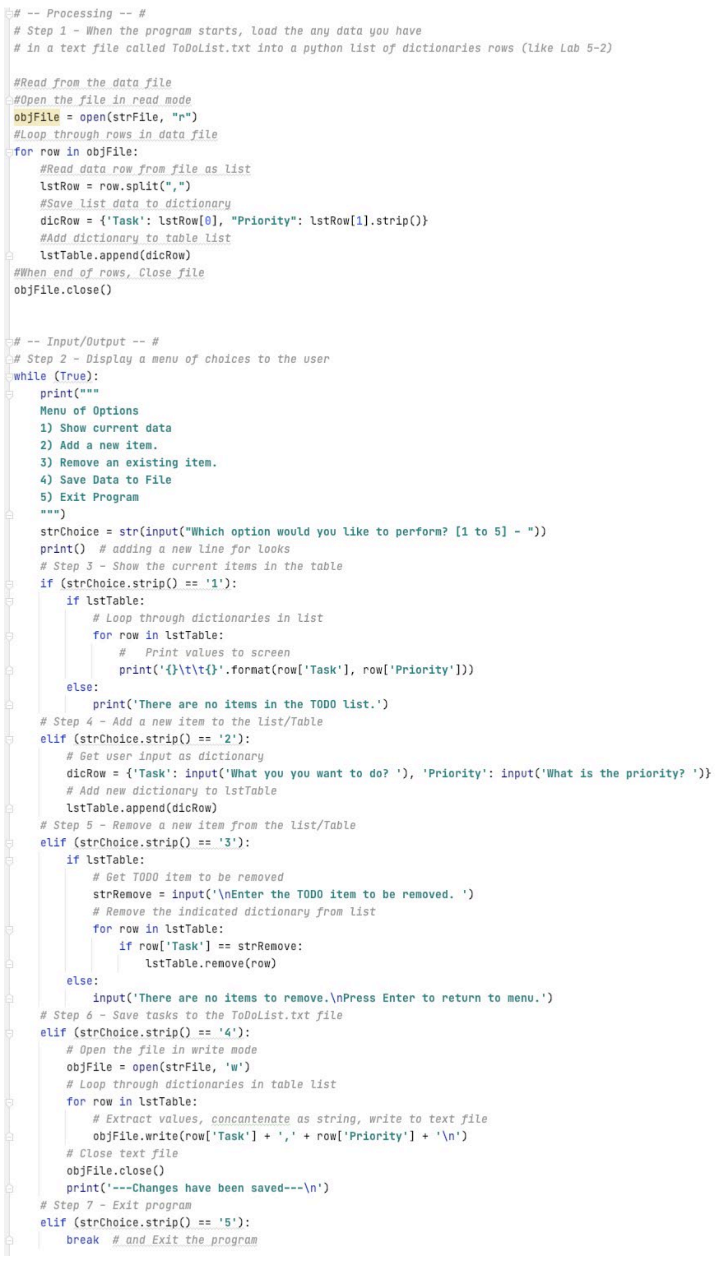

```
elif (strChoice.strip() == '5'):
    break  # and Exit the program
```

Figure 9: Code to exit program

Excluding the variable definitions in the Data section, the code now looks like this.



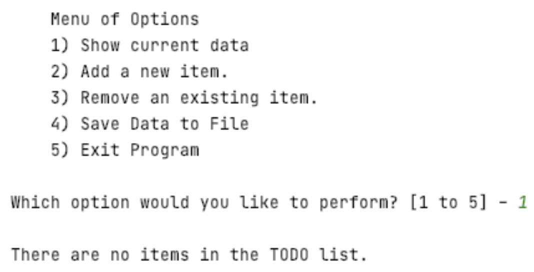
```python
# -- Processing -- #
# Step 1 - When the program starts, load the any data you have
# in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)

#Read from the data file
#Open the file in read mode
objFile = open(strFile, "r")
#Loop through rows in data file
for row in objFile:
    #Read data row from file as list
    lstRow = row.split(",")
    #Save list data to dictionary
    dicRow = {'Task': lstRow[0], "Priority": lstRow[1].strip()}
    #Add dictionary to table list
    lstTable.append(dicRow)
#When end of rows, Close file
objFile.close()


# -- Input/Output -- #
# Step 2 - Display a menu of choices to the user
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] - "))
    print()  # adding a new line for looks
    # Step 3 - Show the current items in the table
    if (strChoice.strip() == '1'):
        if lstTable:
            # Loop through dictionaries in list
            for row in lstTable:
                #   Print values to screen
                print('{}\t\t{}'.format(row['Task'], row['Priority']))
        else:
            print('There are no items in the TODO list.')
    # Step 4 - Add a new item to the list/Table
    elif (strChoice.strip() == '2'):
        # Get user input as dictionary
        dicRow = {'Task': input('What you you want to do? '), 'Priority': input('What is the priority? ')}
        # Add new dictionary to lstTable
        lstTable.append(dicRow)
    # Step 5 - Remove a new item from the list/Table
    elif (strChoice.strip() == '3'):
        if lstTable:
            # Get TODO item to be removed
            strRemove = input('\nEnter the TODO item to be removed. ')
            # Remove the indicated dictionary from list
            for row in lstTable:
                if row['Task'] == strRemove:
                    lstTable.remove(row)
        else:
            input('There are no items to remove.\nPress Enter to return to menu.')
    # Step 6 - Save tasks to the ToDoList.txt file
    elif (strChoice.strip() == '4'):
        # Open the file in write mode
        objFile = open(strFile, 'w')
        # Loop through dictionaries in table list
        for row in lstTable:
            # Extract values, concantenate as string, write to text file
            objFile.write(row['Task'] + ',' + row['Priority'] + '\n')
        # Close text file
        objFile.close()
        print('---Changes have been saved---\n')
    # Step 7 - Exit program
    elif (strChoice.strip() == '5'):
        break  # and Exit the program
```

Figure 10: Processing and Input/Output

Running the program

Executing the program in Pycharm:

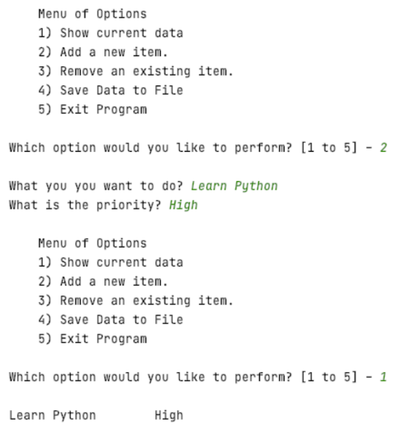
Though it was not shown here, a blank text file was created in the working directory to give the program something to attempt to load data from without throwing an exception. Since the file is blank, when we try to display the contents we get the message that there is no data in the list to display.



```
    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

There are no items in the TODO list.
```

Figure 11: Display data with empty list

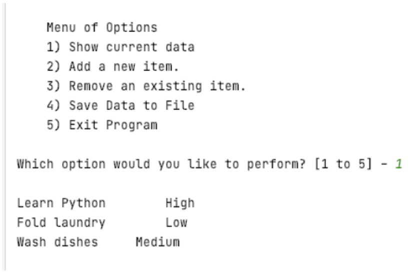
If we add an item, then try the display again, we get the following:



```
    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

What you you want to do? Learn Python
What is the priority? 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

Learn Python        High
```

Figure 12: Add item and display

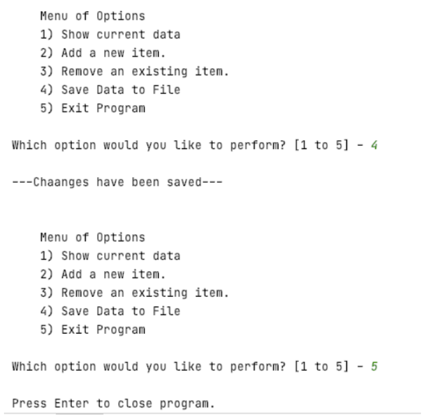
Adding a few more items, we see that the add and display operations appear to be working.



```
    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

Learn Python        High
Fold laundry        Low
Wash dishes     Medium
```

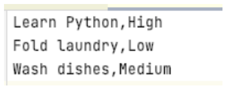
Figure 13: Display with multiple items

Now that we have some data to work with, we will save the data and exit the program, and check the contents of the text file. Then reopen the program and try the display again to see if the data loads.



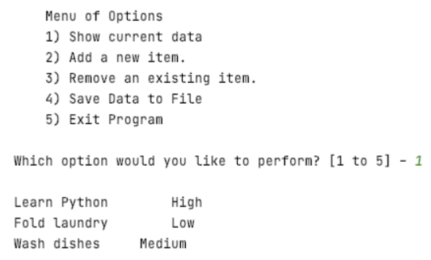
```
    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

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

---Chaanges have been 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

Press Enter to close program.
```

Figure 14: Save and exit



```
Learn Python,High
Fold laundry,Low
Wash dishes,Medium
```

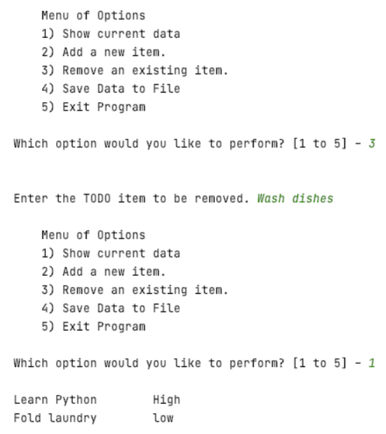
Figure 15: Contents of text file



```
    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] - 1

Learn Python        High
Fold laundry        Low
Wash dishes     Medium
```

Figure 16: Restart and display data

We see that the data was saved, and the program was able to read the file back into memory. Now we need to test removing an item from the list.



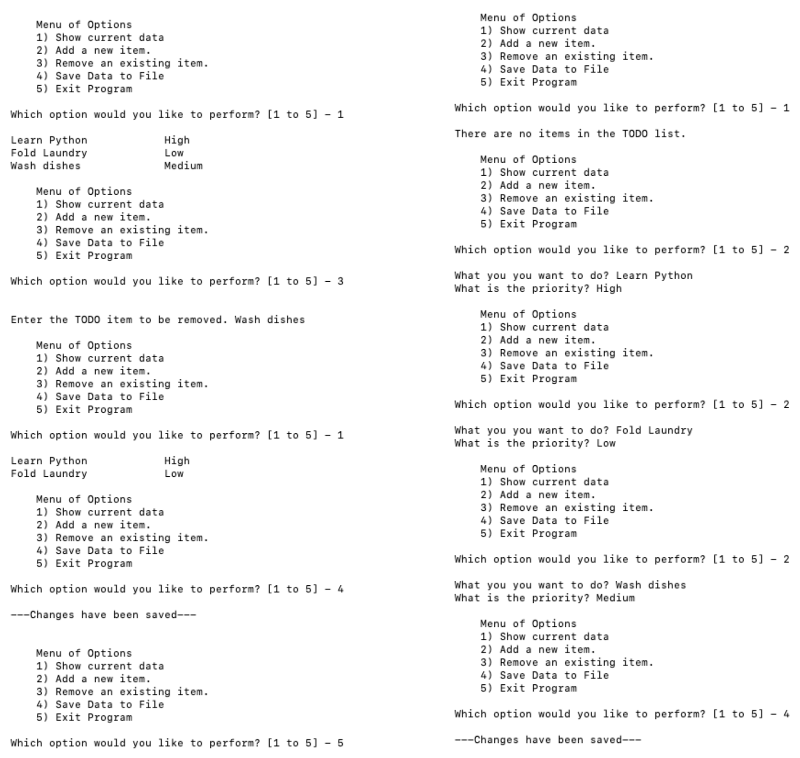
```
    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

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


Enter the TODO item to be removed. Wash dishes

    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

Learn Python        High
Fold laundry        low
```

Figure 17: Remove an item

The removal appears to work, however the way the program is currently constructed - the user must first select to display the list first, to see if what they wanted to delete is in the list. A future upgrade may be to automatically display the existing list when the remove option is selected.

Executing in a Terminal window

As the program appears to be running correctly in Pycharm - we try it in a terminal window, and see similar results.



```
    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

Learn Python            High
Fold Laundry            Low
Wash dishes             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


Enter the TODO item to be removed. Wash dishes

    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

Learn Python            High
Fold Laundry            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] - 4

---Changes have been 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
```

```
    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

There are no items in the TODO 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

What you you want to do? Learn Python
What is the priority? 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

What you you want to do? Fold Laundry
What is the priority? 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] - 2

What you you want to do? Wash dishes
What is the priority? 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] - 4

---Changes have been saved---
```

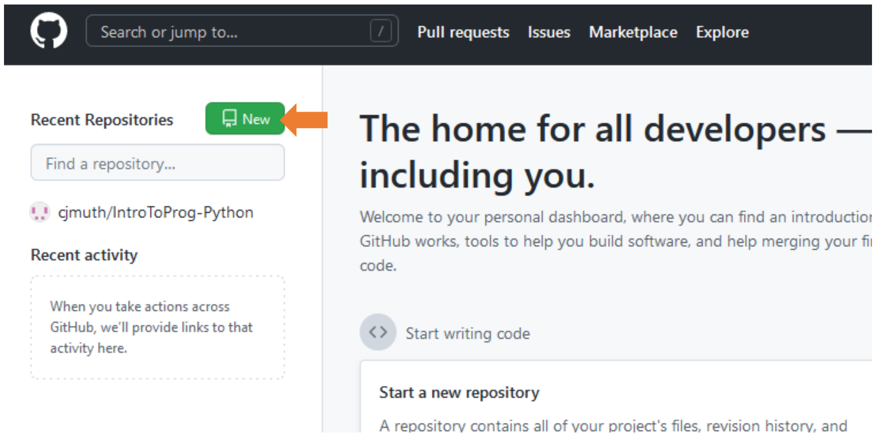
Figure 18: Running in a Terminal window

Post to GitHub

Now that the code is working, the final task will be to post the file to GitHub - a hosting site using version control software original developed by Linus Torvalds for his Linux kernel (McQuiad, Mike. *Git in Practice*. Manning, 2015). Among other things is provides a means to securely store their code off-site, make it available to share with others as needed, and keeping a history of revisions so programmers can compare with or retrieved archived versions of their code.

Instructions for creating an account will not be addressed here, as the website (https://github.com/)(External site) does a fine job of walking the user through the process.

With the account created, the user should be faced with a screen like the one below. In order to start sharing file, they will need to first create repository (a directory/folder on the GitHub site) to store the files by clicking the “New” button.



When you take actions across GitHub, we'll provide links to that activity here.

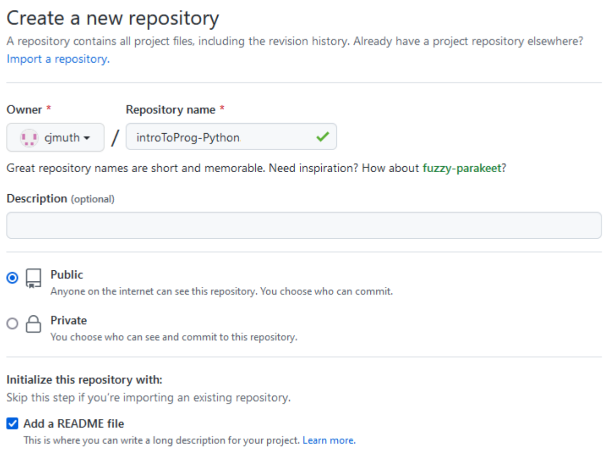
The home for all developers — including you.

Welcome to your personal dashboard, where you can find an introductio GitHub works, tools to help you build software, and help merging your fi code.

A repository contains all of your project's files, revision history, and

Figure 19: Initiate a new repository

The user will then need to assign a name to the repository, and designate it as either Public (available to all GitHub users) or Private (limited access to designated users only). Information on the site recommends that every new repository contains some type of file to insure it's visible – and have provided an option to automatically create a README file for the user.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner *: cjmuth / Repository name *: introToProg-Python ✓

Great repository names are short and memorable. Need inspiration? How about fuzzy-parakeet?

Description (optional)

● Public
Anyone on the internet can see this repository. You choose who can commit.

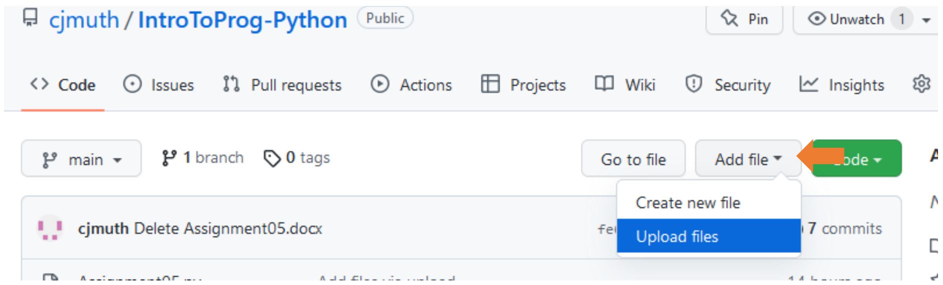
○ Private
You choose who can see and commit to this repository.

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 20: Define attributes of new repository

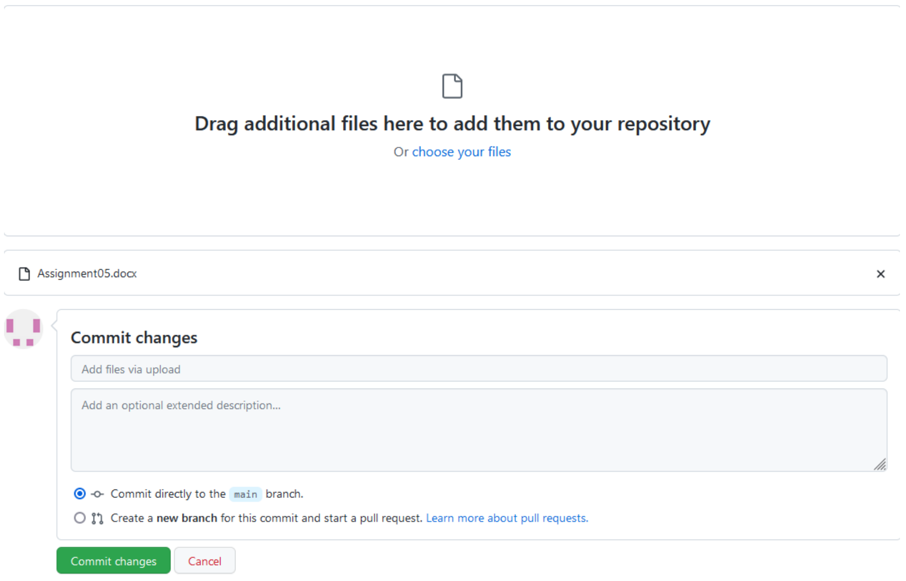
After the new repository is complete, select the "Add file" button to either create new file or upload an existing file.



cjmuth Delete Assignment05.docx

Figure 21: Get files to upload

For this example I chose to upload a file, and in the following window – dragged the Python script from my working folder the drop zone on the webpage, and clicked "Commit changes" to complete the submission. The comments are not required so I left them blank for this project – but for extended projects going through multiple changes they are extremely valuable to identify what changed with each version.


Drag additional files here to add them to your repository
Or choose your files

Assignment05.docx ×

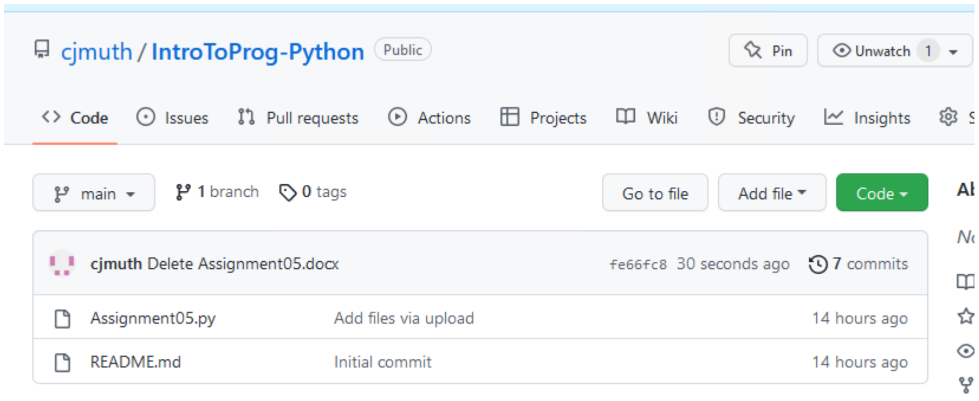
Commit changes

Add files via upload

Add an optional extended description...

[✓ Commit directly to the main branch.]
Create a new branch for this commit and start a pull request. Learn more about pull requests.

Figure 22: Commit – completing the upload


cjmuth / IntroToProg-Python Public

cjmuth Delete Assignment05.docx	fe66fc8 30 seconds ago	7 commits
Assignment05.py	Add files via upload	14 hours ago
README.md	Initial commit	14 hours ago

Figure 23: Upload complete

Summary

In this project we created a program to interact with text file - reading data into memory, modifying it, then writing back to the text file for later use. As the data loaded, it was converted into a set of dictionaries to allow the data to be addressed by keys identifying the elements by name rather than position within the collection.

Although the use of dictionaries instead of lists may not provide a noticeable difference to the users - within the coding environment, the programmer can make use of a dictionary without knowing its detailed structure. Knowing the name of the keys within a given dictionary is enough information to be able to interact with a given element regardless of location. By using meaningful descriptors for keys, the intent of data elements become clearer - making the code easier to understand and maintain.

Finally, the code was uploaded to a GitHub repository to make it available to other users. Looking closely at the screenshots used for the final section, you may notice this document appears the in the repository as well. Since the repository isn't limited to a specific type of file, it can also be used by a single user to store documents or other files they may want to be able to access from multiple computers at different sites (such as home and work) - without concern over losing portable media, or even the computer the work is being developed on.