Dave Phillips

2021-08-02

IT FDN 110 A Su 21: Foundations Of Programming: Python

Module 5 – Assignment 1

So Much To-Do, So Little Time…

# **Introduction:**

In this assignment I’ll be coding a script to keep a basic to-do list for the user. The module will accept three input fields – task, due date, and priority – and dynamically save that information into an external file for later access. Additionally, the user has the option of viewing and/or removing individual saved records.

# **Code Overview:**

The code for this app is split into six major sections. File handling, main loop/input, add records, view records, remove records, and save & exit. The code is written as one giant block as opposed to utilizing functions in order to demonstrate the “long way” to perform actions pertaining to the records.

# **Section 1: File Handle (Figure 1):**

Lines 12 – 25 are where we open (or create) the To-Do List text file. At the beginning of this section I set up two local variables – ‘**mainList**’ and ‘**count**’. It’s important, and much cleaner, to explicitly state these at the beginning this section so I can alter them further down in the code. ‘**mainList**’ will be used to store a list of all to-do list items, including the ones previously saved to the .txt file. ‘count’ will help us determine the current number of records, which in turn allows us to assign sequential ID numbers to each record.

Lines 17 – 22 deal with the initial reading and parsing of the external file. This particular open statement utilizes the “**r**” parameter to signify that we will only be reading the contents of the file. Following the initialization of the file handle, the script loops through each record in the file (delimited by ‘\n’ new line characters), increases the previously defined ‘**count**’ variable by one for each item currently existing in the file. Following that, we use the **split** method to separate each extracted line into a list of individual components before finally appending that list to the **mainList**. I was having a bit of trouble with data being deleted from the file prematurely, so I’ve opted to completely close the file handle before reopening it under the “**w**” parameter on line 25.

1. # File handler
2. mainList = []
3. count = 1
4. # Read file and extract current contents
5. fhandle = open("ToDoList.txt", "r")
6. for record in fhandle:
7. count += 1
8. recordList = record.split(",")
9. mainList.append(recordList)
10. fhandle.close()
11. # Re-open file for writing (with 'append' parameter)
12. fhandle = open("ToDoList.txt", "w")

Figure 1. Initial file handling

# **Section 2: Collect Initial User Input**

**Lines 27 – 53 concern prompting the user for their initial input. To start I define the variable running on line 28 which I can then use to create an infinite loop that will continue asking the user for their choice until they decide to exit the script. This is beneficial in the sense that the individual won’t need to restart the app on the command line each time they want to take an action.**

**Following that, the main prompt is located on lines 30 – 40. This is one single print statement but broken up over multiple lines. This is especially useful when formatting is important as you can more easily see and manipulate where the text sits on a particular line.**

**Lines 42 – 51 contain a simple try/except block that filters out any input by the user that *doesn’t* fall within the accepted range of numbers (in this case 1-4). This is done in two steps. First, the program checks to see if the value entered by the user can be converted to an integer (line 44). If the entry can be converted, the next line then checks to see if the number is \*not\* within the acceptable range. This setup is more prudent since we can use a continue statement to return to the top of the loop as opposed to nesting any large chunks of code underneath the value checks that were just performed. (Figure 2)**

1. **try:**
2. **user\_input = int(input("Please select an option. [1 to 4] || "))**
3. **if user\_input not in range(1, 5):**
4. **print("Invalid Input!")**
5. **print("Here")**
6. **continue**
7. **except:**
8. **print("Invalid Input!")**
9. **continue**

**Figure 2. Checking for valid input**

# **Section 3: Adding Records**

Lines 56 – 94 have to do with adding records to the file – yeah, this is a long section. We’ll break it down bit by bit. Lines 56 – 60 request the basic task information from the user. The **add\_id** is an automatically determined number based on the count value derived from lines 14 & 19 (which I mentioned would make sense a bit farther down in the code). This makes it easy to search for and delete specific records if the user selects that option from the main menu. Assigning ID numbers to data tables is generally good practice anyway. Following the ID number the script requests data on the task name and due date. A good option at this point would be to convert the date into a date object using python’s built-in ‘**time**’ module, but I didn’t think of that at the time of coding (and didn’t want to go back and change it! Ha!).

I’ve added an additional feature that tracks the priority of each to-do item (because I overachieve far too often). To this end there’s an additional multi-line print statement giving the user a rundown on what each priority level is.

Like the **try/except** block in the main menu loop, the code on lines 70 – 82 simply assures that the user has put in an acceptable value for the ‘priority’ parameter. Similar logic applies as before. Finally if everything checks out properly, line 82 saves the values to a list object for easy storage when writing the final set of records to the external file.

Additionally, I’ve added a step where the user my confirm or reject their input in case something incorrect was entered by mistake. This uses a simple f-string method to insert the variables directly into a print statement for the user to see. They have the option to confirm and if their input is “y” or “yes” (line 89), the new record is appended to the **mainList** object, which again, will be written to the external file as the final step. As we are using the value of ‘**count**’ as the id number “tracker”, it’s important to increase that value as new records are added (otherwise all new records will have the same id number, which of course defeats the purpose).

Lastly, line 94 send the user back to the “main menu” to continue the process.

# **Section 4: Viewing Records**

Similar to the previous iteration of this script, the option to view records is rather simple. Line 103 goes through each record currently contained in the **mainList** and, on line 115, **print**s out each record in turn in a pre-formatted style. The code on lines 104 – 112 is purely for aesthetics when displaying the data, as it only substitutes the priority number with the associated priority name. Again, this is to avoid any confusion from only having numerical values with no reference when displaying the records. (Figure 3)

1. # Apply actual name to "Priority" field for display
2. for row in mainList:
3. priority = int(row[3])
4. if priority == 1:
5. priority = "High"
6. elif priority == 2:
7. priority = "Medium"
8. elif priority == 3:
9. priority = "Low"
10. else:
11. priority = "Oops..."
12. # Display record
13. print(row[0] + ". | " + row[1] + " | Due: " + row[2] \
14. + " | Priority: " + priority)

Figure 3. View records with priority word substitution

# **Section 5: Removing Records**

The longest portion of code, removing records is a bit more straightforward that one might expect; particularly when using the ID number as a reference.

To delete a record the script will prompt the user for a record ID (line 124). This can be found by browsing the current list with the View Records option, which is outlined in the above section. The code on lines 128 – 136 check to see if the user has chosen to exit without deleting a record or if a number was provided with a simple **try/except** statement as there was above.

Lines 139 – 144 loop through the records in **mainList**, which is to say all the records, searching for a match to the integer entered by the user. If a match is found, the requisite variables are set to their corresponding values and have the string equivalent of their priority number applied for easier viewing (also as described above).

Lines 158 & 159 perform the same function as lines 85 & 86 in the Add Record section – confirmation. Again, using an f-string, we’re able to display the record called by the user and confirm deletion.

Lines 161 – 167 are simply input handlers for the potential “yes” or “no” responses provided by the user. If the action is confirmed we use the **pop** method of the list class to remove the item at a specific location within that list. In this situation, we simply take the record ID provided by the user and subtract one since list indices start at zero, but ID numbers start at one. (Figure 4)

1. if confirmation in ("y", "yes"):
2. mainList.pop(int(record\_id) - 1)
3. elif confirmation in ("n", "no"):
4. continue

Figure 4. Removing record from master list

**Section 6: Saving the Data**

Lastly, the final lines in the code (lines 170 – 177) perform a simple **write** task to the external file. In this particular situation, the entirety of the file is overwritten each time, but since all data is stored in the **mainList** object, it can be looped through and written quickly and cleanly using the formatting on line 174. (Figure 5)

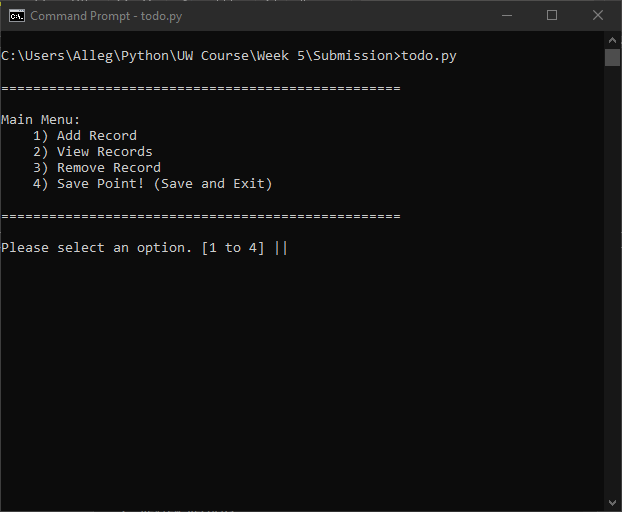
1. for item in mainList:
2. fhandle.write(item[0] + "," + item[1] + "," + item[2] + "," .....

Figure 5. Save to file

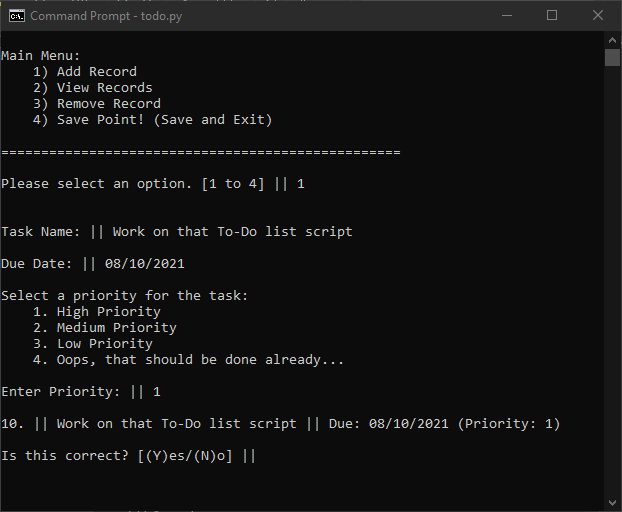
# **Summary:**

The difficulty in this assignment is how best to keep code clean and organized, in my opinion. Since the use of functions was explicitly prohibited, there is a significantly higher amount of “reused” code, that is, code that has been typed in more than once. Additionally, the main program loop is huge which makes troubleshooting a bit more difficult when something fails.

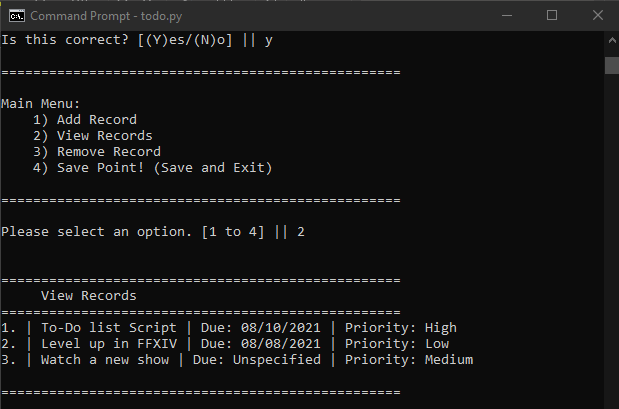
Each section, as outlined above, is represented in the following appendices.



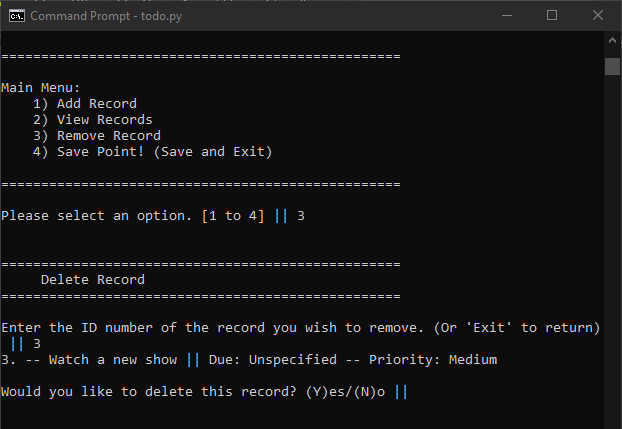
Appendix A. Initial menu via command line



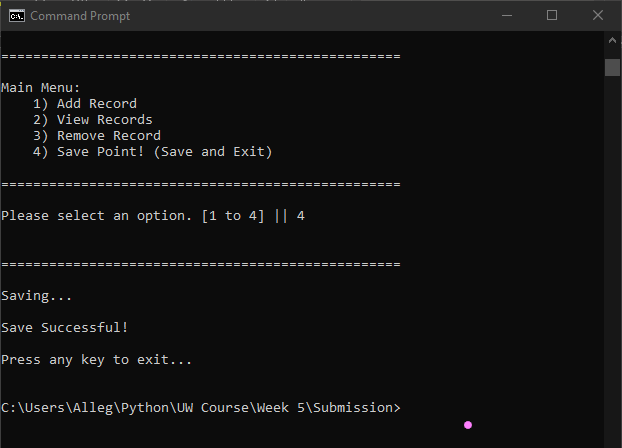
Appendix B. Add Record



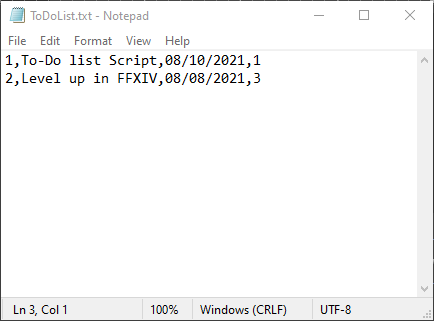
Appendix C. View Records



Appendix D. Remove Record



Appendix E. Save and Quit



Appendix F. Raw .txt file