Thomas Nord

08-03-2022

Foundations of Programming (Python)

Recap of Module 05 – Lists & Dictionaries

# [**https://github.com/nordthomas/IntroToProg-Python**](https://github.com/nordthomas/IntroToProg-Python)

Creating a To Do List Program

# Introduction

This week we’re creating a To Do List program. We will need to be able to capture use input about what task they want to add and its priority. We will save that data to a table and need to be able to save that table to a file. Once we have a file with data, we should be able to take that data and add it to a dictionary before converting to our table. This week we will also be looking at the ability to remove a row of data from our table. Let’s get started.

# Setting up our variables and pseudo code

Before we begin coding it always makes sense to sit down and consider what we want our program to do and how we want to achieve those goals. We should consider what actions the user can perform then consider the areas of concern we need to make that happen. Once we understand those items then we can begin to put together some pseudo code. And finally, we can assemble an initial batch of variables.

## Pseudo Code

The best place to start is with pseudo code. As that has been provided for us this week in the Assigment05\_Starter.py file we don’t have a lot to plan for this step. The pseudo code is broken down in to 3 areas of concern:

# -- Data -- #  
# -- Processing -- ## -- Input/Output -- #

In the **Data** section we’ll set up our variables. In the **Processing** section we’ll look to see if a text file exists. If there is an existing text file, we’ll read the data from it into our working memory and if there isn’t a text file already, we’ll create one. Then in the **Input/Output** section we’ll add our program’s menu and central logic.

## Declaring our Variables

For our variables we know we’ll need an external file to read from/write to (strFile & objFile). Per our instructions for this assignment, we’ll need to capture the data in the file to a dictionary (dicRow) then convert that data to rows (lstRow) and build a table (lstTable) using lists. We’ll be capturing a user’s menu choice (strChoice), as well as the Tasks (strTask) they want to add and the task’s Priority (strPriority). We’ll also need a variable later to capture the row the user wishes to delete (intDelete) if they choose that menu item. Most of these variables we simply want to declare and not assign a value currently. However, we do know that we want our text file to be created in the same directory as the script and have it named “ToDoList.txt” so we can assign that to strFile now.

# Processing our Data

Before we present the user with our menu we need to first see if there is any existing data that we’ll need to utilize. If there isn’t a text file already we’ll need to create one. If there is data, then we’ll build a table that we’ll use throughout the rest of the program.

To check if there is already a file we set up a conditional. It checks to see if objFile has a value of None and, if so, we know there isn’t an existing document and need to create one. We call the open function and add strFile as an argument for the path. Because we need to create a file we need to use “w” (write) or “a” (append) for the mode. Though it won’t make much difference in terms of memory usage we go ahead and close the file when we’re done. Our code ends up looking like this:

if objFile == None:  
 objFile = open(strFile, "a")  
 objFile.close()

If there is already a file with some data we need to read that data in so we’ll start with the same open function but set “r” (read) for the mode. To read the data we need to set up a for loop to perform some actions based on each row. Because I have made the choice later in the program to separate Task and Priority with a pipe (|) I also need to split the row so we only capture the task/priority and not the separator.

objFile = open(strFile, "r")  
for row in objFile:  
 lstRow = row.split(" | ")

We’re off to a good start and now we have a basic list. If we were to print out the list as it appears at this point it would look like this:

['Do a thing', 'Now\n']

The assignment wants us to go a step further and convert this list row to a dictionary. In order to do that we need to assign some keys and give them some values.

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

The .strip() at the end will remove the \n we have at the end of our second list item. If we printed dicRow, we would see:

{'Task': 'Do a thing', 'Priority': 'Now'}

Rather than a list where the index for each item is an implied number starting at 0 for the first item, we have a key for each index and we assign a value to that key. All we need to do now is add this dictionary row to our table with the .append method. Adding items from a dictionary row is very much like adding items from a list, you just have to remember that instead of a numbered index you are calling the dictionary’s key:

lstTable.append(dicRow["Task"] + " | " + dicRow["Priority"].strip())

We add the .strip method at the end to remove the hidden new line escape character. Don’t forget to close your file with the .close method!

# Input/Output

Now that we’ve set up a place for our data to go, we need to collect it. Like last week we’ll start with a menu of user options inside of a while loop:

# -- 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]: "))

Our strChoice variable uses input to capture the user’s choice and convert it to a string.

## Option 1 – Show Current Data

Our logic flow starts with an if/else loop to evaluate the user’s input against the different menu options.

# Step 3 - Show the current items in the table  
if (strChoice.strip() == '1'):

Here the user chooses option 1 so we need to show any data we may have in our table. Because there may not be data in the table yet I decided to add a check for that. I do this by comparing the length of lstTable against 0. If they are equal, I print a message to the user.

if len(lstTable) == 0:  
 print("There are no tasks in the list.")

If that statement is not True then I proceed to the else statement which prints a simple header for my to do list and then uses a for loop to print each row of lstTable. The continue statement returns us to the main menu after the table has been displayed.

else:  
 print("Task | Priority")  
 for row in lstTable:  
 print(row)  
continue

## Option 2 – Add a New Item

The 2nd option needs to be able to add a new item to lstTable.

# Step 4 - Add a new item to the list/table  
elif (strChoice.strip() == '2'):

We have already done this with the Home Inventory program so we can make quick work of this section. First, we need to gather our data from the user. We’ll need them to enter a task and it’s priority so we call two of the variables we declared earlier (strTask & strPriority) and use the input function to take user input. We’ll reuse lstRow to store that information.

strTask = input("Enter a Task: ")  
strPriority = input("Set a Priority: ")  
lstRow = [strTask, strPriority]

Once we have a list it’s a simple matter of using the .append method to add it to lstTable. I added a message to the user to confirm their task was added to the list and then the continue statement brings us back to the main menu.

lstTable.append(lstRow[0] + " | " + lstRow[1])  
print("\nTask added to list.")  
continue

## Option 3 – Remove an Existing Item

The 3rd option in the menu is perhaps the trickiest.

# Step 5 - Remove a new item from the list/table  
elif (strChoice.strip() == '3'):

It seems simple enough to remove a row from our table, but first we need a way to display the table for reference and to tell the user the appropriate method for indicating that row. For my design I chose to number the rows and ask the user to indicate which numbered row should be removed.

i = 0   
print("Task | Priority")  
for row in lstTable:   
 i += 1   
 print("(" + str(i) + ")" + " " + row)   
intDelete = int(input("\nWhat row number would you like to delete? "))

To start we set a counter (i) to 0. This will be used to help us number our table’s rows. The next line is the header row for our list. We follow that up with our standard for loop to print the rows but with a twist. Every time we iterate through the table we’re going to increment our counter and then print that value in our string. The string shows the counter value concatenated with the row data to produce an easy to parse list. Our last step for this section of the code is to ask the user to indicate which row they would like to delete and capture that as a variable.

The next section of the code is meant to validate the user’s input. If the number the user selects matches a number of one of our list’s rows, delete that row. If the number does not match an existing row number, we ask the user to select a valid number.

if intDelete >= 1 and intDelete <= len(lstTable):  
  
else:  
 print("Please enter a valid number.")  
 continue

There are two issues with this approach I was not able to solve:

1. I don’t validate against non-numeric characters.
2. An invalid choice will take the user back to the main menu rather than asking the user to re-enter a number.

I hope to learn more about how to solve these issues in the coming weeks.

If the user’s entry is a valid row number we pass that in to our delete (del) statement as the index for our lstTable list. Because the list we presented the user begins with 1 and lists are zero-based we need to reduce the intDelete value by 1 in order to delete the correct row.

Next, I added some feedback for the user. I set up a conditional based on the presence of items in lstTable. If, after deleting their selected row, there are no items remaining we tell the user as much. If there are still items in the list after the removal we tell them that their row has been deleted and present them with the newest iteration of the list. Once we have completed that task we return them to the main menu.

del lstTable[intDelete-1]   
if len(lstTable) == 0:  
 print("\nThere are no tasks in the list.")  
else:  
 print(f"\nRow {intDelete} has been deleted.\n")   
 print("Task | Priority")  
 for row in lstTable:   
 print(row)  
continue

## Option 4 – Save Data to File

The 4th option is another one we’ve seen before with the Home Inventory program. We’ll take the table that we’ve built and write it out to the text file we created earlier. The first thing we need to do it re-open the file with the open function.

# Step 6 - Save tasks to the ToDoList.txt file  
elif (strChoice.strip() == '4'):  
 objFile = open(strFile, "w")

I used write (“w”) mode here because we want to be sure to overwrite any existing content in the text file. Using append (“a”) would put the current table at the end of the document and since we read in the data from this document earlier to create our table we would likely be duplicating some information.

We use a familiar for loop to iterate through the rows of our table and use the write function to write that data to our external file. We have to remember to add the new line escape character (“\n”) or all of our data would be on the same line and would look terrible.

for row in lstTable:  
 objFile.write(row + "\n")

With the data written we just have some quick clean up items to take care of. First, we’ll close the file now that we’re done writing to it, then print a message to the user letting them know we’ve written the data to their ToDoList.txt file and use the continue statement to send the user back to the main menu.

objFile.close()  
print(f"Tasks saved to {strFile}.")  
continue

## Option 5 – Exit Program

The 5th option allows the user to exit the program. A simple break statement was provided in the starter file. While this would have been good enough I added a simple print statement to bid the user adieu.

# Step 7 - Exit program  
elif (strChoice.strip() == '5'):  
 print("Goodbye.")  
 break

# Testing our Code

The last step of the coding process is to test our code. We’ll run some basic tests to make sure our code is working in each of the areas of concern. We don’t have any user-facing elements for our Data area but we’ll see through the testing of the other areas that our variables are working as expected.

The first area of concern is Processing. You’ll recall this is where we checked to see if a document was already available and, if not, we create one. If there is a document we read in that data to our table (Figures 1.1 and 1.2).

Text

Description automatically generated

Figure 1.1: Our document had no data

Text

Description automatically generated

Figure 1.2: Our document had data

Now let’s add some data. We’re asked to put in a task and its priority. The program let’s us know when we’re done that the task has been added to the list (Figure 2).

Text

Description automatically generated

Figure 2: New task added

Next up is deleting an existing item from our list. We get a list of the items currently available in the list and they’ve been indexed for better usability. The program then confirms the selected row has been deleted and shows us an updated list (Figure 3).

Text

Description automatically generated

Figure 3: Removing an item

With our list complete it’s time to save it back to our file. Selecting option 4 writes the data to the external file and confirms the save is complete (Figures 4.1 & 4.2).

Text

Description automatically generated

Figure 4.1: Save complete

Graphical user interface, text, application

Description automatically generated

**Figure 4.2: Data in our file**

Finally, let’s make sure our program exits properly (Figure 5).

Text

Description automatically generated

Figure 5: A clean exit

# Summary

This week we built on existing knowledge about reading from and writing to text files. We learned how to convert our list rows in to dictionary rows, then to use those dictionary rows to build a table. And finally, we learned how to build a menu that allows a user to manipulate that table data by adding and removing rows.