

# ToDo List Program

1	INTRODUCTION	1
2	CHECKING FOR AN EXISTING TODOLIST & LOADING DATA	1
3	INPUT / OUTPUT SECTION	2
3.1	Show current items	2
3.2	Adding a new item	3
3.3	Removing an item	4
3.4	Writing the table back to the file and ending the program	5
4	SUMMARY	6

## 1 Introduction

This write-up will cover the steps involved and the Python fundamentals used to manage a ToDo list. A starter Python script was presented as a basis for beginning the project. The program should begin with loading each row of data from a text file named ToDoList.txt into a list of dictionary objects. Each row in the list will be a dictionary with a key/value pair. The program will prompt the user to select options to view the tasks in the table, add or remove an item from the table, and finally to write the table back to the file.

## 2 Checking for an existing ToDoList & loading data

The first step is to load any data in the ToDoList.txt file into memory. To ensure the file exists, a try-except error handling method is used. Try-except tries a section of code, and if it runs without error, it omits the except section. If it sees an error, it will execute the code in the except section. This is useful because it allows the programmer to control what the user sees if an error occurs. The first step in loading the data in the text file to memory is to use the open() function and pass the file name and mode parameter. Next, a for function will look through each row, split each string of text by a comma and save each element in a list. Now a list object

in memory with 2 elements, can be saved to a dictionary row, using the curly bracket symbols, and passing the 0<sup>th</sup> and 1<sup>st</sup> element of the list to the dictRow variable, along with the key/value designations of 'task' and 'priority' being hard-coded. Next, each dictRow is appended to an empty list Table using lstTable.append(). Finally, the lstTable is printed to the console so show all the dictionary key/value pairs as elements in the list. See Listing 1. Each row has a key/value pair for the task, and a key/value pair for the priority in Figure 1.

```
# -- Processing -- #
# Step 1 - When the program starts, load any data you have
# in a text file called ToDoList.txt into a python list of dictionaries rows
# (like Lab 5-2)
try: # try to load data from ToDoList.txt into a list of dictionary rows
    objFile = open('ToDoList.txt', 'r') # open ToDoList.txt in read mode
    for row in objFile: # for each row in objFile
        strData = row.split(",") # assign comma separated data to strData,
        # split by ","
        dicRow = {'Task': strData[0], 'Priority': strData[1].strip()} # load
        # each strData list element into a dicRow
        lstTable.append(dicRow) # append lstTable with each dicRow
    objFile.close() # close the file
except: # if the ToDoList.txt file doesn't exist - make a txt file
    objFile = open("ToDoList.txt", 'w')
    objFile.close()
    print("ToDoList.txt not found. Creating a new ToDoList.txt file")
```

*Listing 1. Try-Except and data loading section*

```
/usr/local/bin/python3.9 "/Users/ryanlupinski/PycharmProjects/_PythonClass/Module05 - Lists and
Dictionaries/Assignment05/Assignment05_Starter.py"
{'Task': 'Task', 'Priority': 'Priority'}
{'Task': 'Cook', 'Priority': '1'}
{'Task': 'Clean', 'Priority': '2'}
{'Task': 'Vacuum', 'Priority': '3'}

Process finished with exit code 0
```

*Figure 1. Task key / value and priority key / value*

Figure 1 shows the columns of tasks and their priority. Each row of the lstTable is a dictionary with the task and priority. Lists are different from dictionaries in that they are more flexible and elements of many different kinds can be stored, but dictionaries need a key/value pair where the key must be a string and the associated value can be strings, lists, tuples, constants, etc. List elements are accessed with an index while dictionary elements are accessed with a key.

## 3 Input / Output section

### 3.1 Show current items

Now that any existing data has been loaded into memory, the user is presented a menu of options to show current data, add a task, remove a task, save data to file, or exit the program. Beginning with option 1: show current data, the current items in the table will be displayed row by row in Listing 2

```
# Step 3 - Show the current items in the table
if strChoice.strip() == '1':
    for row in lstTable: # for each row in lstTable
        strData1 = row['Task'] # Use 'Task' key to get task
        strData2 = row['Priority'] # Use 'Priority' key to get priority
        print(strData1 + " | " + strData2) # print the row separated by a
bar
    continue
```

*Listing 2 Show current data in the file*

This returns each dictionary row in the lstTable so the user can see what items are currently in the table. The format is cleaned up and the task and priority are printed and separated by a vertical bar, see figure 2.

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

task | priority
cook | 1
laundry | 3
dust | 1
gym | 8
pay bills | 2
```

*Figure 2 Task and priority key/value are printed to console*

### 3.2 Adding a new item

To add a new item, the lstTable must be appended with a new dictionary row. The user will be prompted to enter a task and a priority. Each task and priority is saved as a string. A new dictionary row is created with the task and priority. The keys 'Task' and 'Priority' are hard-coded for continuity.

```
# Step 4 - Add a new item to the list/Table
elif strChoice.strip() == '2':
    strTask = input("Enter a task: ") # prompt user for the task
    strPriority = input("Enter the priority: ") # prompt user for the
priority
    dicRow = {'Task': strTask, 'Priority': strPriority} # create a new
dicRow with new task
    lstTable.append(dicRow) # append the table w/ new dicRow
    continue
```

*Listing 3 add a new task and priority into the table*

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

Enter a task: sweep
Enter the priority: 1
```

*Figure 3 Adding a new task and priority*

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

task | priority
cook | 1
laundry | 3
dust | 1
gym | 8
pay bills | 2
sweep | 1
```

Figure 4. Output showing new task and priority being added

We can see that the task to sweep has been added to the lstTable with a priority of 1.

### 3.3 Removing an item

To remove an item, the user is prompted to enter a task that is store in memory with the variable removeChoice. Then, the program loops through the rows using the range and length function on the lstTable. If the specific row's task key is equal to the removeChoice task, the specific row of the lstTable is deleted and the if function breaks. See listing 4.

```
# Step 5 - Remove a new item from the list/Table
elif strChoice.strip() == '3':
    removeChoice = input("What task do you want to remove? ") # prompt user
    for task to delete
        for row in range(len(lstTable)): # look in each row of the lstTable
            if lstTable[row]['Task'] == removeChoice: # check if task key equals
the removeChoice
                del lstTable[row] # delete the row
                break # break the if loop
        continue
```

Listing 4 Remove task from lstTable

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

What task do you want to remove? dust
```

Figure 5 Remove task 'dust'

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

task | priority
cook | 1
laundry | 3
gym | 8
pay bills | 2
sweep | 1
```

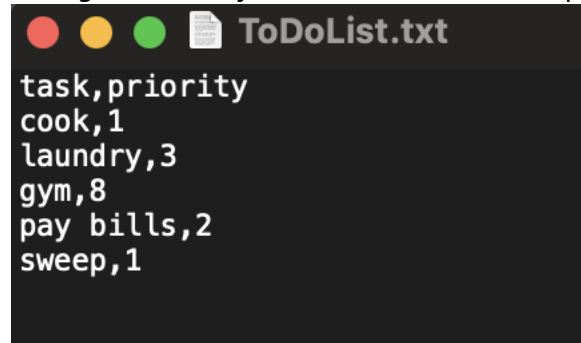
Figure 6 Dust is removed from the lstTable

### 3.4 Writing the table back to the file and ending the program

The final step is writing the `lstTable` of tasks and priorities back to the file. The user selects option 4 to save data to the file or option 5 to exit the program. A connection to the file is opened in write mode and each row of the `lstTable` is looped through. Two variables `strData1` and `strData2` call the keys 'Task' and 'Priority' and saves the respective value. The task and priority are written to the file, separated by a comma, and a new line is added.

```
# Step 6 - Save tasks to the ToDoToDoList.txt file
elif strChoice.strip() == '4':
    objFile = open('ToDoList.txt', 'w') # open connection to file in write
mode
    for row in lstTable: # for each row lstTable
        strData1 = row['Task'] # pull task value from lstTable
        strData2 = row['Priority'] # pull priority value from lstTable
        objFile.write(strData1 + ',' + strData2 + '\n') # write
task/priority values to file
    objFile.close()
    continue
# Step 7 - Exit program
elif strChoice.strip() == '5':
    print("Goodbye")
    break # and Exit the program
```

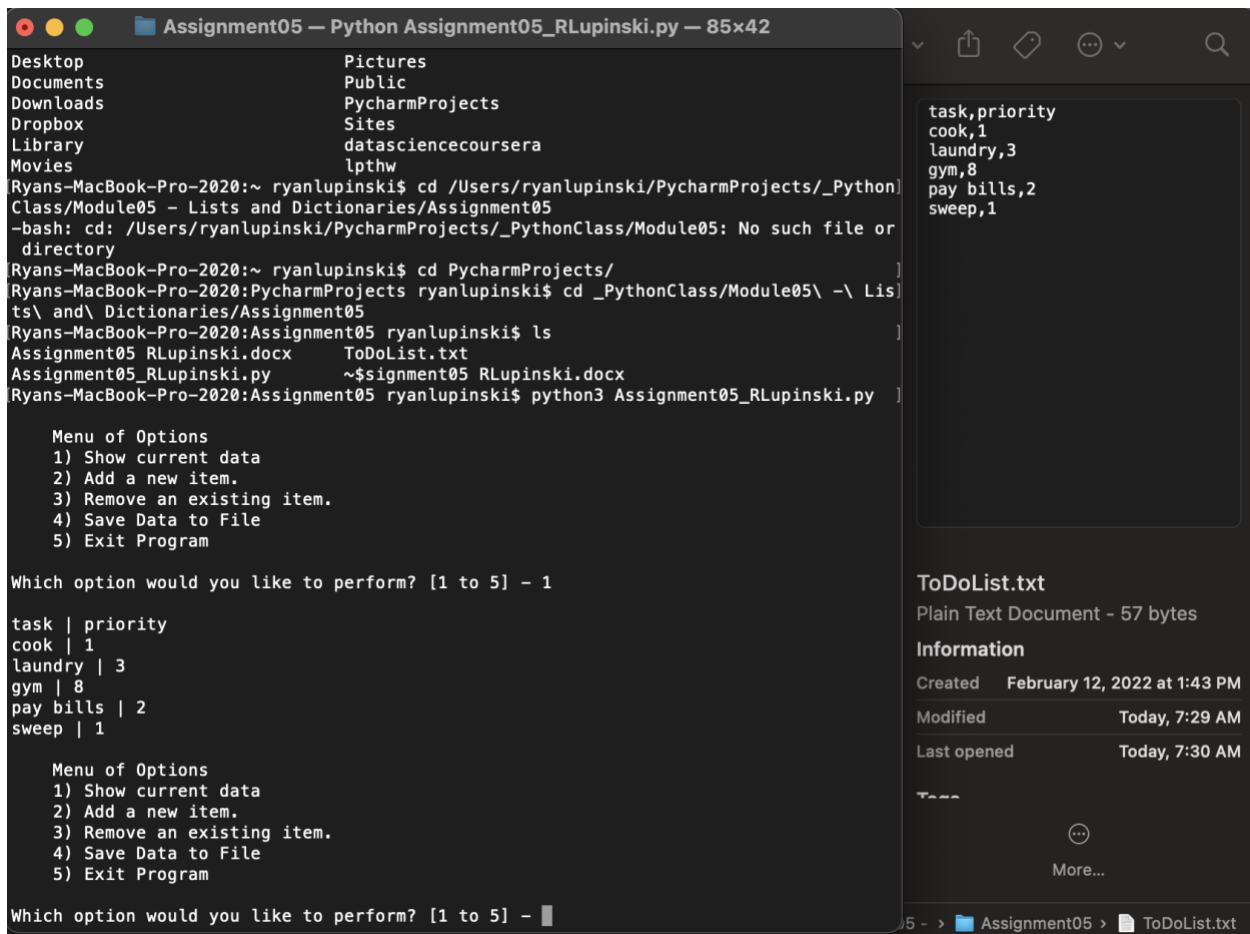
*Listing 5 Write to file code section and end program sections*



The screenshot shows a text editor window with a title bar containing three colored circles (red, yellow, green) and a document icon, followed by the text 'ToDoList.txt'. The window's content area has a dark background and displays the following text in a light-colored monospace font:

```
task,priority
cook,1
laundry,3
gym,8
pay bills,2
sweep,1
```

*Figure 7 lstTable written to file*



```
Assignment05 — Python Assignment05_RLupinski.py — 85x42
Desktop      Pictures
Documents   Public
Downloads    PycharmProjects
Dropbox      Sites
Library      datasciencecoursera
Movies       lpthw
Ryans-MacBook-Pro-2020:~ ryanlupinski$ cd /Users/ryanlupinski/PycharmProjects/_PythonClass/Module05 - Lists and Dictionaries/Assignment05
-bash: cd: /Users/ryanlupinski/PycharmProjects/_PythonClass/Module05: No such file or directory
Ryans-MacBook-Pro-2020:~ ryanlupinski$ cd PycharmProjects/
Ryans-MacBook-Pro-2020:PycharmProjects ryanlupinski$ cd _PythonClass/Module05\ -\ Lists and Dictionaries/Assignment05
Ryans-MacBook-Pro-2020:Assignment05 ryanlupinski$ ls
Assignment05_RLupinski.docx  ToDoList.txt
Assignment05_RLupinski.py    ~$signment05_RLupinski.docx
Ryans-MacBook-Pro-2020:Assignment05 ryanlupinski$ python3 Assignment05_RLupinski.py

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
cook | 1
laundry | 3
gym | 8
pay bills | 2
sweep | 1

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] -
```

task,priority  
cook,1  
laundry,3  
gym,8  
pay bills,2  
sweep,1

**ToDoList.txt**  
Plain Text Document - 57 bytes

**Information**

Created	February 12, 2022 at 1:43 PM
Modified	Today, 7:29 AM
Last opened	Today, 7:30 AM

More...

Figure 8 ToDo List program running in a terminal session

## 4 Summary

Reading and writing to a file is an important skill in Python. In this program, the existence of a file is checked with try-except, data is loaded into memory with the open() function, tasks and their priority can be added to the lstTable in memory, the current data can be printed in a clean format, and the data can be written back to the file with the write() function. An important aspect of programming is the separation of concerns. This program's code has different sections that define variables, process data, and an input/output section. This is important for visibility and a best practice to help understand what needs to be done.