M Bruce

Aug 8, 2022

IT FDN 110 (B)

Assignment 05

Link: <https://github.com/Jane2024/IntroToProg-Python>

Dictionaries and File I/O in Python

# Introduction

This week covered “dictionaries” and how they handle collections of data in Python, as well as how they compare to “lists”. The following pages summarize those topics and what was learned in storing and loading data to and from files with example work from the labs. The last section will cover how this week’s assignment 05 was coded, its display output and I/O to files and what was learned.

# Lists

In Python, as learned in previous assignments, a list is a collection of data, which can hold different types of data, and uses a unique syntax to create it -brackets define a list “[ ]”. There are a number of built-in functions that can be used to manipulate list data. Some of these handle input and output of list data from files. If a list’s data is to be persisted, then it needs to be placed in a file to maintain it after the program is closed, since the data lives in memory while the program is running and is cleared when the program closes.

## Storing list data to file

The “open()” function can be used to handle writing list data to file. The first argument is the name of the file, the second argument is the action to be taken- “w” write, “a” append. Below is an example code segment in ***Listing 5.0*** and its output in ***Figure 5.0*** that demonstrates storing list data to a file, where a row shows the top 2 songs of the week, the artist and the song name.

# Variables to be used

lstRow = []

strFileName = 'StoredData.txt'

objFile = None

# Storing data to file

objFile = open(strFileName, "w")

lstRow = ["Pos#", "Artist", "SongTitle"]

objFile.write(lstRow[0] + '\t' + lstRow[1] + '\t'\*2 + lstRow[2] + '\n')

lstRow = ["1", "Charley Puth", "Left and Right"]

objFile.write(lstRow[0] + '\t' + lstRow[1] + '\t' + lstRow[2] + '\n')

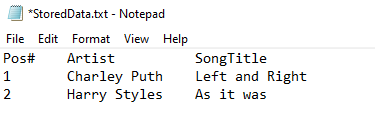
lstRow = ["2", "Harry Styles", "As it was"]

objFile.write(lstRow[0] + '\t' + lstRow[1] + '\t' + lstRow[2] + '\n')

objFile.close()

Listing 5.0 Code segment

The output of the above code segment when stored to file is in figure 5.0

  
Figure 5.0 “StoredData.txt” file content

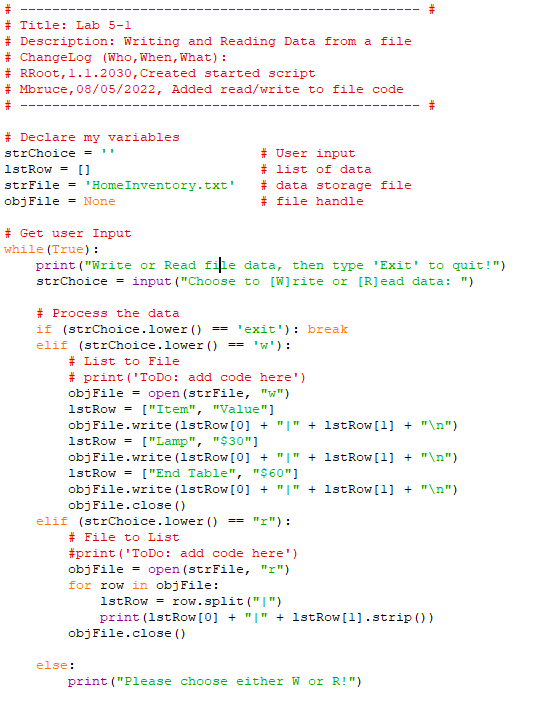
## Loading list data from file

Loading data from a file back into memory, allows a program to access and use the data.

Two functions are commonly used to separate elements in the list and to strip unwanted carriage returns when reading from files.

## Lab 5-1

This lab allows a user to choose to read or write data from/to a text file using a “while” loop, the user has three options to choose: read, write or exit. It then enters an if/elif conditional statement testing for the entered user input. If the user chooses to write to file, then the provided data examples are entered and a new text file is created with the new data. If uses chooses to “read” file then it loads the file output from HomeInventory.txt to memory and displays it. Else the user exits the program. The revised code to accomplish this is seen in Listing 5.1 below.



Listing 5.1 code segment

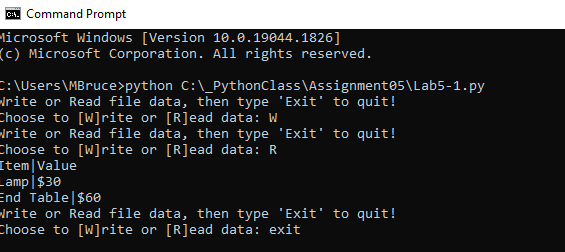
*The output of this is seen in figure 5.1  
*

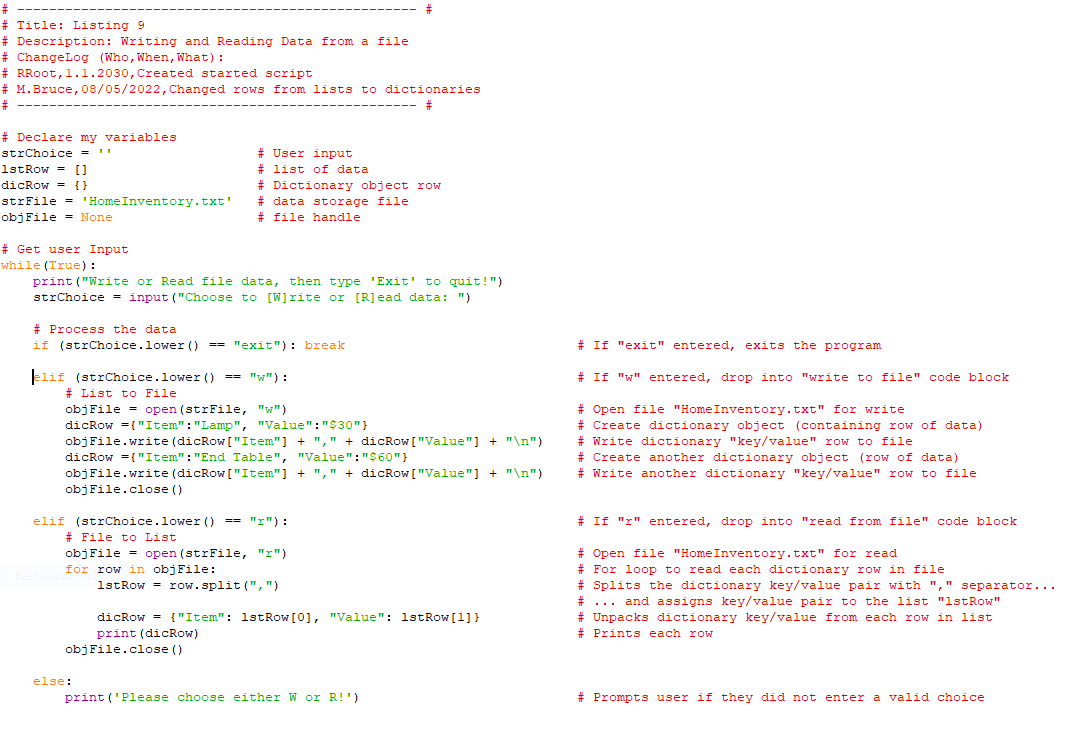
Figure 5.1 Lab5.1 - cmd prompt output

# Dictionaries

**Dictionaries** are another data collection type. They can hold different data types, but store their data in Key/Value pairs. Instead of using numerals for the index like a list, the index subscripts are **key (character) subscripts** and stores a corresponding value. The braces {} operator is used to create a dictionary**.**

## Lab 5-2 Working with a Table of Dictionaries

For Lab 5-2, after defining two string variables, a list, a dictionary and a file object, the program starts with a while loop and prompts the user for a desired action to either write to file or read from file using file object methods: open(), write(), close(). Or they can exit the program. See **Listing 5.2** for Lab5-2 code segment.



Listing 5.2 **Lab5-2 script code segment**

In testing the script, the home inventory items and their values are written and read from file. In the original script version, the items and values are stored in a ***list object*** and written to and read from a file. In the modified version for Lab 5-2, they are separated into ***dictionary key/value objects***, with the item name associated with dictionary key “Item”, and the item’s value associated with VALUE. The dictionary objects are then stored as a row in a list table. When read, the process is reversed, each row is processed and split using a “comma” for separator. Then the list of dictionary objects are output to display.

In **figure 5.2** below, the Lab5-2 script code listed above is run and output in the cmd prompt.

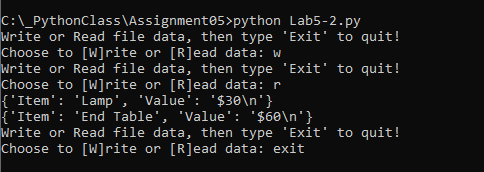
****

Figure5.2 **Lab5-2 output in cmd prompt**

The output file ***HomeInventory.txt*** content , which lab5-2 created is seen in **figure 5.3 below**.

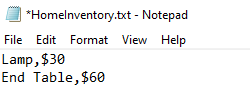
****

Figure5.3 **Lab5-2 file - HomeInventory.txt**

# Improving scripts

There are four simple techniques that can be used to improve a script’s readability and maintainability. They are: **separation of concerns programming patterns, functions, script templates, and structured error handling**.

Separation of concerns programming patterns

A design principle for separating an application or program into distinct sections is called separation of concerns. It is a natural fit for object oriented or modular programming, which aligns with defining the operations of a program within objects or modules, with each one handling a specific operation. The principle is similar to encapsulation, where functionality is isolated within a defined classes or functions. In most programs, sections can usually be divided into three different sections; Data, Processing, and Presentation (or Input-Output). Some benefits of this practice are code re-use, saving on programming for an operation multiple times and easier code maintenance.

Functions

Functions group a set of statements or a code segment that handle a particular operation under a single name. It is called with specified arguments. Functions need to be defined before they are used and this is best done at the start of the script in a section for defining functions just as when defining variables to be used in a program. When a function call is reached in the Main body of the script, the program skips up to the function and executes its statements, then returns to the main body right after function call.

Script templates

Templates help provide consistency in programming scripts and helps maintain a more readable structure. Many IDEs offer the ability to create and recall templates when programming scripts.

Structured error handling

No matter how well designed a program is, there will be errors. Planning for them by using good error handling in Try-Except blocks helps provide a clear customized message of what went wrong, and allows the program to fail gracefully or continue running.

# Asignment05

This assignment was to modify an existing script template that manages a "ToDo” list containing two columns of data, "Task" and "Priority." The data is loaded into Python Dictionary objects, representing a row of data, which is then added to a list object tocreate a table of data,

Create Script  
1) Opened PyCharm and created folder and project named “Assignment05”

2) Added script named **Assignment05\_starter.py** to project at:  
 *C:/\_PythonClass/Assignment05/* **Assignment05\_MBruce.py**

3) Updated header info, and define variables to be used in the program

## 1st step- Load contents of ToDo.lst

Opened file for read and iterated through list rows to load into memory. Below is the code segment that handles this step in ***Figure 5.4***

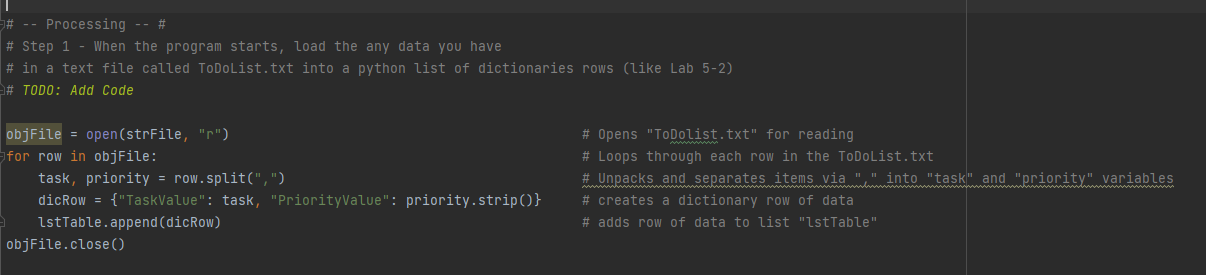


Figure5.4 **Assignment05 step 1 code segment**

## 2nd step- Display menu

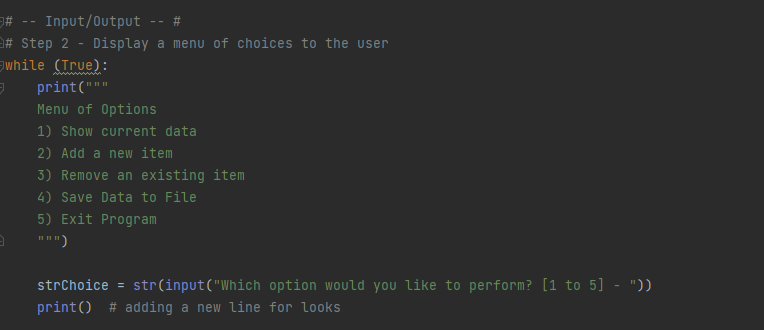
The menu display was already handled as part of the starter code, so no new code work had to be done. Below is the code that handles the menu display in ***Figure 5.5***.

Figure5.5 **Assignment05 step 2 code segment**

## 3rd step- Show the current items in the table

If menu item # “1” was chosen, a loop was created to iterate through the rows in the list table and print out the values of each of the directory objects. ***Figure 5.6*** shows the code for this operation.

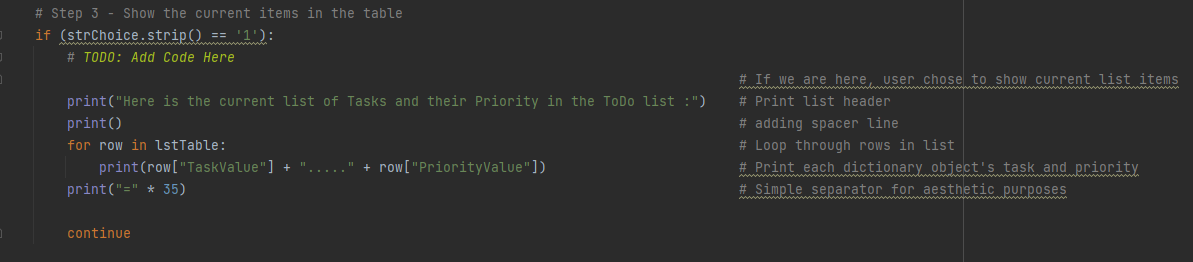


Figure5.6 **Assignment05 step 3 -code segment**

The output of the current items in the list table when the program is run is seen below in ***Figure 5.7***.

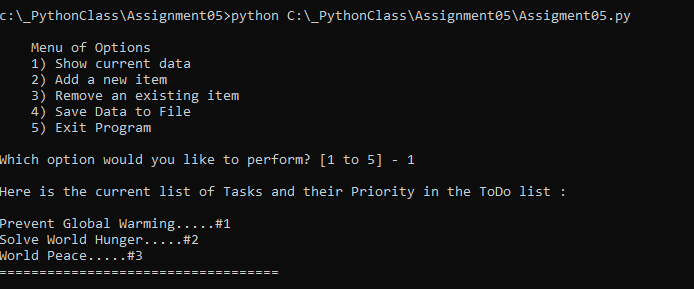


Figure5.7 **Assignment05 step 3 output in cmd prompt**

## 4th step- Add new item (task) to list table

If menu item # “2” was chosen, then it prompted and took in two string variables for a new task and its priority. Afterwards, it stored the values in the list table as a new directory entry. Then it would prompt the user for next desired action. User could continue to add tasks until choosing to stop. The code segment for this is shown in ***Figure 5.8*.**

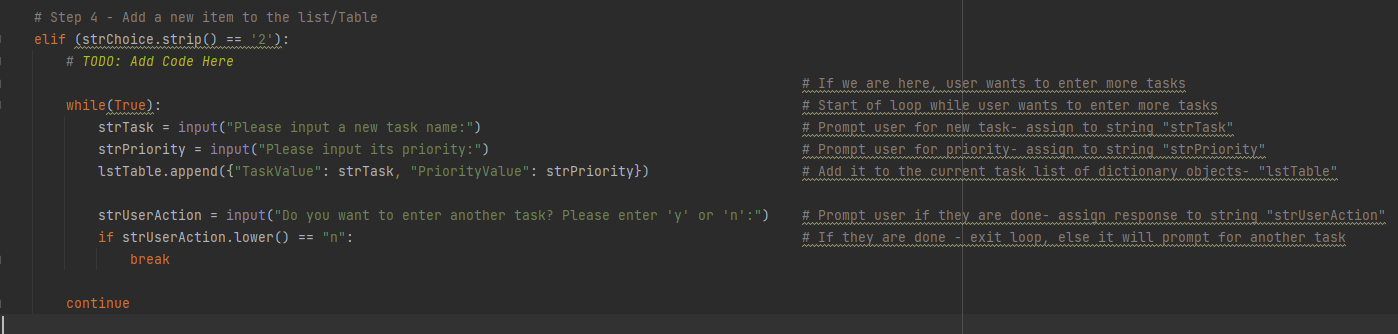
****

Figure5.8 **Assignment05 step 4 -code segment**

In **figure 5.9** below, the output in cmd prompt is seen when adding a new task “Conserve water use” is added to list of tasks

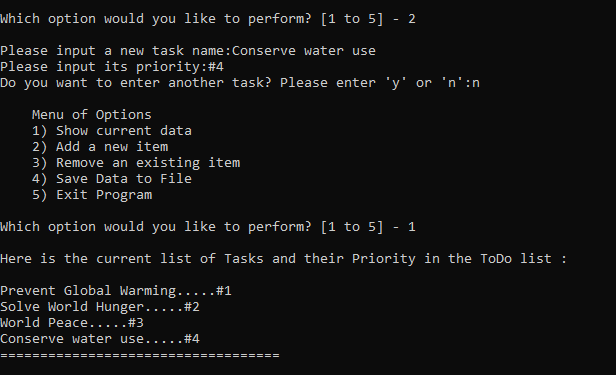


Figure5.9 **Assignment05 step 4 -output in cmd prompt**

## 5th step- Remove item (task) from list table

If menu item # “3” was chosen, then user is prompted to enter the name of the task to be removed from current list. Matching this user entry to a value in the list table proved to be a challenge for me when it came to using for loops and checking the list specifically for the task value and displaying results of search and removal. After some research, I found “list comprehension” worked nicely for this. It easily handled recursing through all the values and placing them in a list, which allowed for an easy check for a match of the entry to any current TaskValue in the list.

Here is a comparison of what “lstTable” contents are and what “lst\_of\_all\_values” are.

***lstTable printout-***  
*[{'TaskValue': 'Prevent Global Warming', 'PriorityValue': '#1'}, {'TaskValue': 'Solve World Hunger', 'PriorityValue': '#2'}, {'TaskValue': 'World Peace', 'PriorityValue': '#3'}]*

**lst\_of\_all\_values printout-**

*['Prevent Global Warming', '#1', 'Solve World Hunger', '#2', 'World Peace', '#3']*

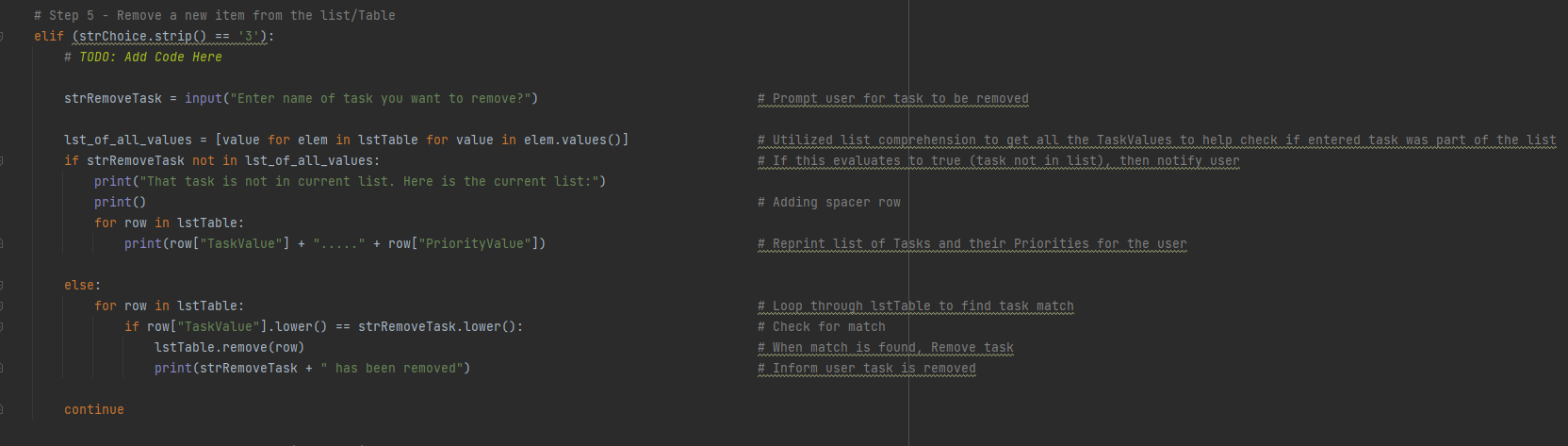
See figure 5.10 below for code segment of this step’s operation.

Figure5.10 **Assignment05 step 5 -code segment**

In figure 5.11 below, the task “Conserve water use” was removed when selecting option #3 in menu.

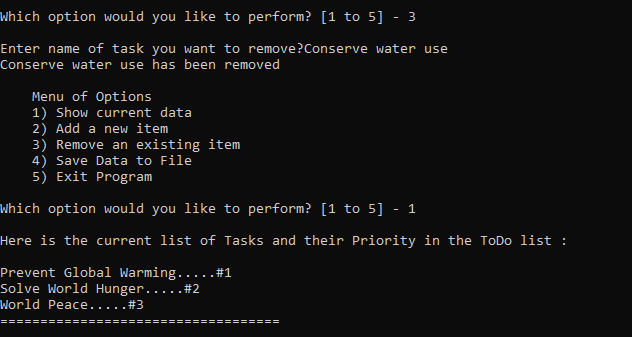


Figure5.11 **Assignment05 step 5 -output**

## 6th step- Process the Data into a File

This step allowed the user to save the current list to the ToDoList.txt file. It just needed to iterate through the list table and write each of the dictionary objects (task and priority values) to the file. See **Figure 5.12** below for code segment.

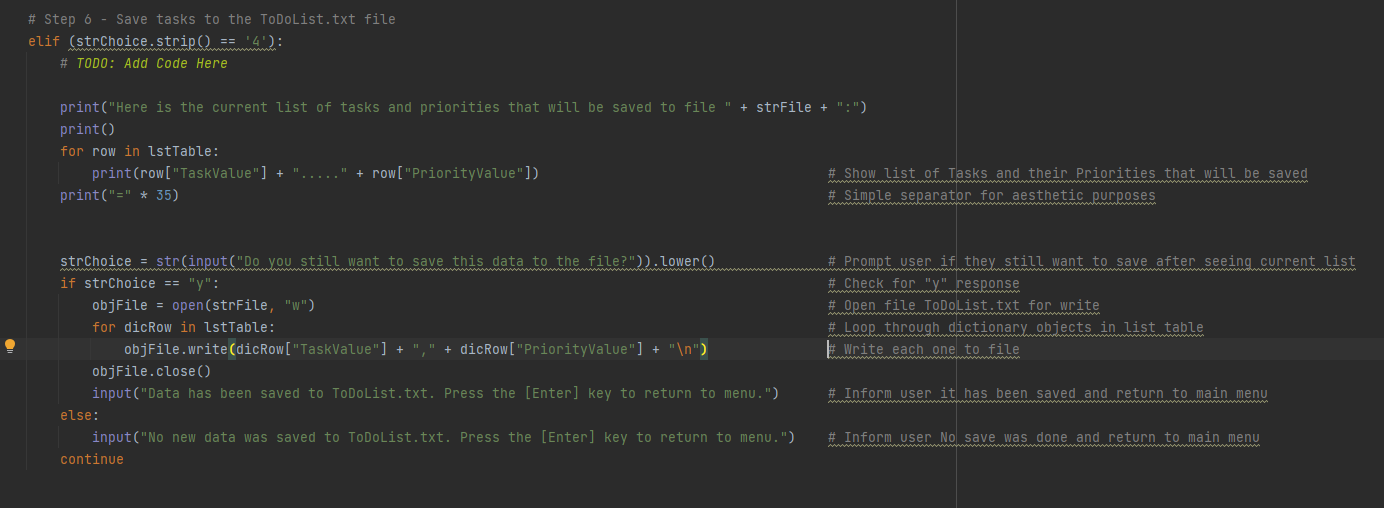


Figure5.12 **Assignment05 step 6 -code segment**

The output when selection option #4 to save the list data to file is captured in ***Figure 5.13*** below. If the user selects “n”, then list is not saved to file and program returns user to main menu.

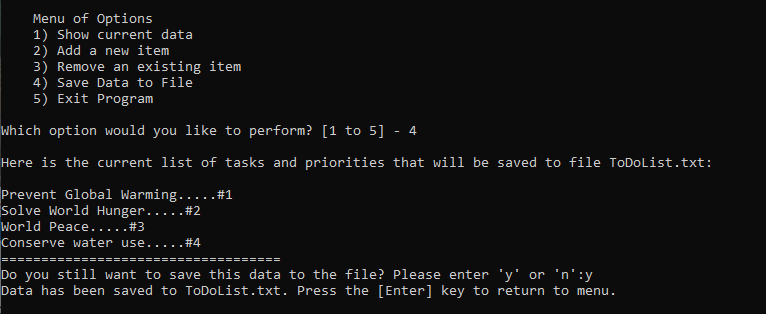


Figure5.13 **Assignment05 step 6 -output**

In ***Figure 5.14***, the contents of the ToDoList.txt file is seen after saving the list contents to file.

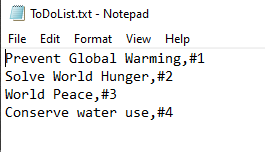


Figure5.14 **Assignment05 step 6 -saved file contents**

## 7th step- Exit program

If user selects option # 5, the program exits. User informed program is exiting. See output in **Figure 5.15**

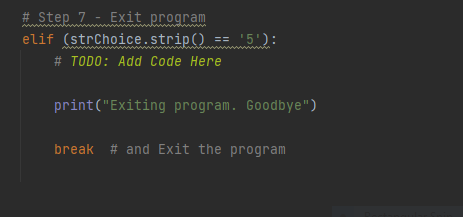


Figure5.15 **Assignment05 step 7 – code segment**

# Summary

In the module05 course video, readings, and web links, the focus was on dictionaries and file input and output.

Learning to iterate through lists of directory objects was challenging. Seemed like a simple concept, but actually searching the list for the directory objects, finding matches and printing out contents of the list table was a bit difficult. There was a lot of redundant tasks for listing of contents, which should be helped once we can use functions and employ better code reuse.