Name: Stephen Jenkins

Date: August 12th, 2023

Course: FDN 110 A: Foundations of Programming – Python

Assignment: Asgmt-05

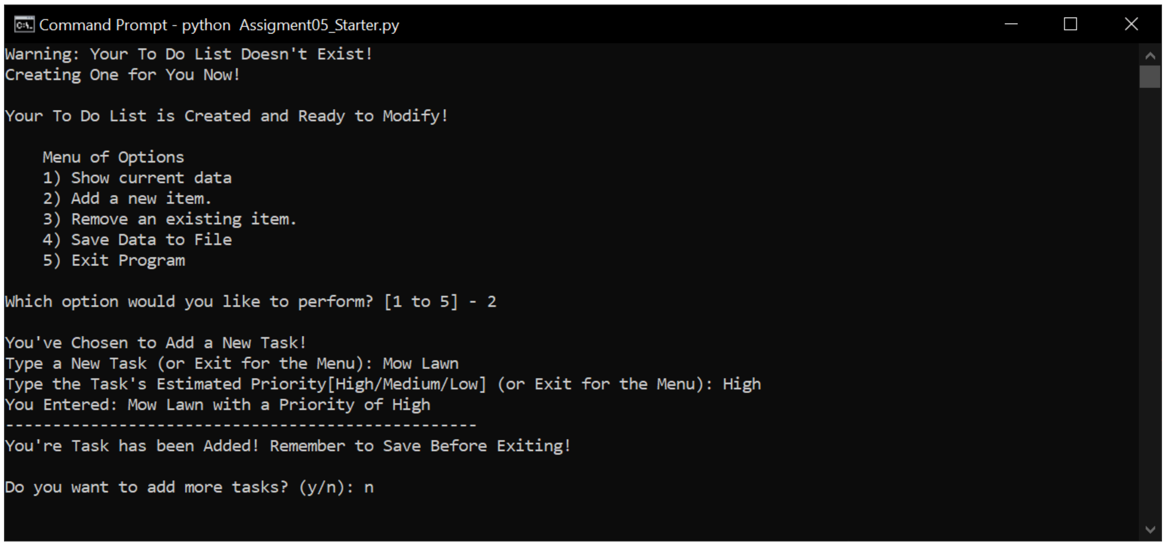
The To Do List Application?

Introduction

This week's assignment for Module Five of the "Foundations of Programming – Python" class continued the previous week's theme of storing and accessing data. Up to this point students have focused more on Tuples and in a more limited capacity Lists. To continue this trend of exploring further the concept of Sequences, or collections of data, they were tasked with creating an application where users would input a chore or "To Do" task and assign it a priority. Then, they could either view the data or write that data out to an external text file. To achieve this, Dictionaries were introduced, which allow for the storage of data, the same way Lists and Tuples do but using a set of keys, rather than location indexes. These dictionaries can then be nested within Lists and accessed.

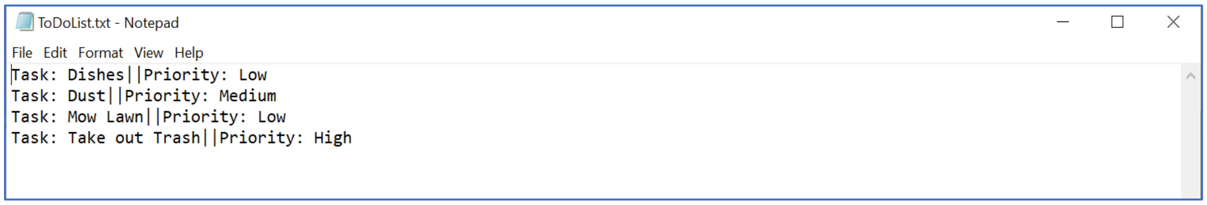
This being the fifth week of the class, students are being introduced to more realistic coding scenarios, such as, having to reverse-engineer someone's code, and continue the work. Randal Root provided, within the assignment, a script which he started, leaving key functionality out of the equation to see how students would adapt and finalize the code for a complete end-to-end experience.

As a final step for this assignment an additional tool, that is used heavily programming, was introduced, allows students to store and link to their assignment assets. This tool is called Git, a repository-based source control program that allows for the storage, versioning, and collaboration of script files and programs. Git has become a valuable tool for those looking to grow their teams and increase collaborative efforts as teams grow, and become more spread around the world.



```
Warning: Your To Do List Doesn't Exist!
Creating One for You Now!

Your To Do List is Created and Ready to Modify!

    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

You've Chosen to Add a New Task!
Type a New Task (or Exit for the Menu): Mow Lawn
Type the Task's Estimated Priority[High/Medium/Low] (or Exit for the Menu): High
You Entered: Mow Lawn with a Priority of High
--------------------------------------------------
You're Task has been Added! Remember to Save Before Exiting!

Do you want to add more tasks? (y/n): n
```

Figure 1: The desired result of this assignment



```
Task: Dishes||Priority: Low
Task: Dust||Priority: Medium
Task: Mow Lawn||Priority: Low
Task: Take out Trash||Priority: High
```

Figure 2: Data written to text file

NOTE: Not every combination has been QA'ed. If used in order, as the assignment dictated, the functionality works as intended.

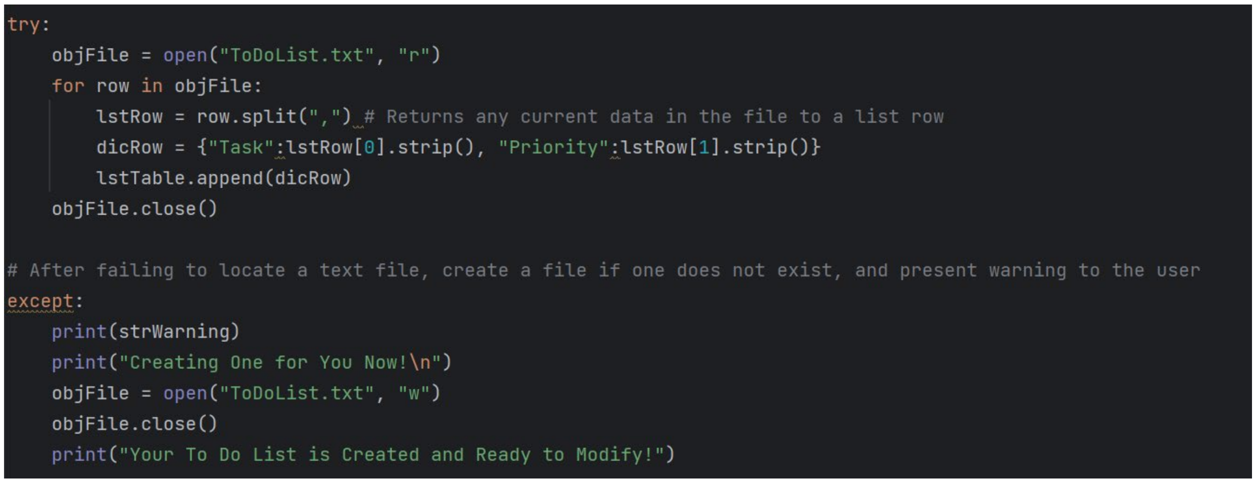
Required Resources and Tools

As has been covered in the previous weeks there were new tools and resources taught that provided additional assistance in creating and executing the weekly assignment. Some of the new concepts are covered in more detail below, with examples from the final, written assignment. Lists will not be covered in this write-up since they were detailed in the previous week's turn-in.

Try & Except – Error Handling

There are times when the programmer may not be able to anticipate how a user will interact with their product or have accommodated for every potential fault. To help reduce issues with the user experience there are introduced methods of "Error Handling". One that was implemented in this week's assignment was "Try & Except", which allows a series of organized statements to be attempted, if they cannot be completed, instead of throwing an error or crashing, the script can move on to a second set of statements, and either print a warning or run additional functionality.

For this assignment the "Try" looked for an existing "ToDo" file, a text document. If it was found it would read the data into dictionaries and save the rows out to a table. If a document could not be found and read, it would let the user know and create one, readying it for use.



```
try:
    objFile = open("ToDoList.txt", "r")
    for row in objFile:
        lstRow = row.split(",") # Returns any current data in the file to a list row
        dicRow = {"Task":lstRow[0].strip(), "Priority":lstRow[1].strip()}
        lstTable.append(dicRow)
    objFile.close()

# After failing to locate a text file, create a file if one does not exist, and present warning to the user
except:
    print(strWarning)
    print("Creating One for You Now!\n")
    objFile = open("ToDoList.txt", "w")
    objFile.close()
    print("Your To Do List is Created and Ready to Modify!")
```

Figure 3: Try & Except statements from the Assignment

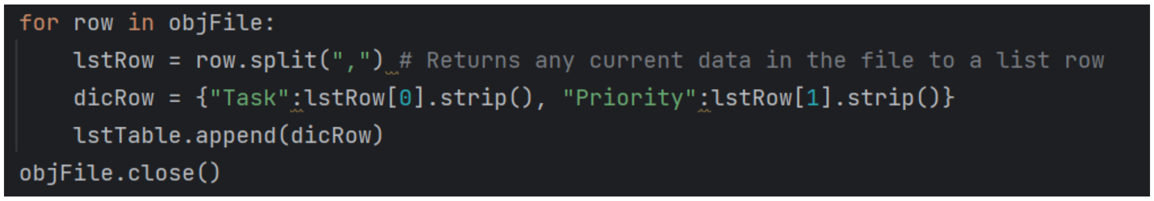
Dictionaries

Dictionaries are another type of Sequence within Python that allows the storage of data, that looks like an array, or List. However, unlike Tuples and Lists, Dictionaries are not referenced by their index, or subscript location within the Sequence. Each Value has a paired Key that can be used to find, reference and write the value with. This can come in handy when storing personal information, for example; "Name" would be the Key, and "John Smith" would be the value.

Dictionary data is stored with the Key and Value being displayed next to each other, separated by a colon ":". The entire Sequence is encapsulated by curly brackets "{}", versus the Tuple that uses parentheses "()", and the List that uses regular brackets "[]".

Dictionaries can be stored or nested within a List or Tuple. Within the completed assignment a Dictionary was created that stored a "Task" as the Key and an assigned "Priority" as the Value. These were then stored within a List, structured as a Table.

dicExample = {"Name":"John Smith", "Phone":"Mobile"}



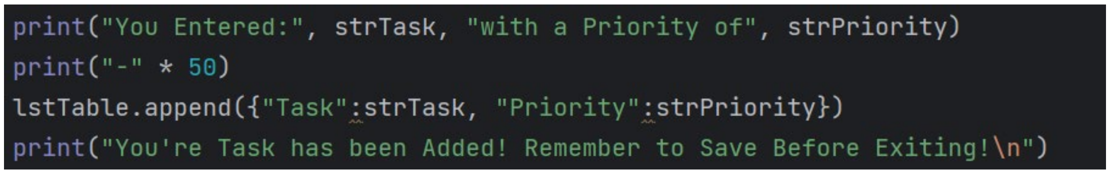
```
for row in objFile:
    lstRow = row.split(",") # Returns any current data in the file to a list row
    dicRow = {"Task":lstRow[0].strip(), "Priority":lstRow[1].strip()}
    lstTable.append(dicRow)
objFile.close()
```

Figure 4: Dictionary, stored in a Table, from the Assignment

Append Function

The Append "append()" function allows for objects to be added, attached, or supplemented to the endx of an existing list.

The assignment required the students to store rowed data in a table and allow users to either add or remove items from that table. With the Append function data could be entered, and then tacked on to the end of the existing table, created at the beginning of the application. This table was revisited many times during the user experience.



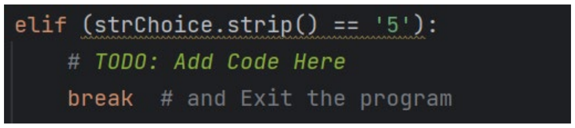
```
print("You Entered:", strTask, "with a Priority of", strPriority)
print("-" * 50)
lstTable.append({"Task":strTask, "Priority":strPriority})
print("You're Task has been Added! Remember to Save Before Exiting!\n")
```

Figure 5: Example of the Append function being used on a List Table.

Strip Function

During the input and formatting of varying data, whitespaces can be introduced in leading or trailing areas of statements, or strings. To ensure clean, consistent storage the Strip "strip()" function can be used to remove these extraneous whitespaces. It can also be used to remove any character added as an argument within the function.

Strip was used within the turned-in assignment to ensure any trailing whitespaces or carriage returns were removed so that formatting stored out to the text document was correct.



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

Figure 6: Example of the Strip function within the Assignment

Breaking Down the Application

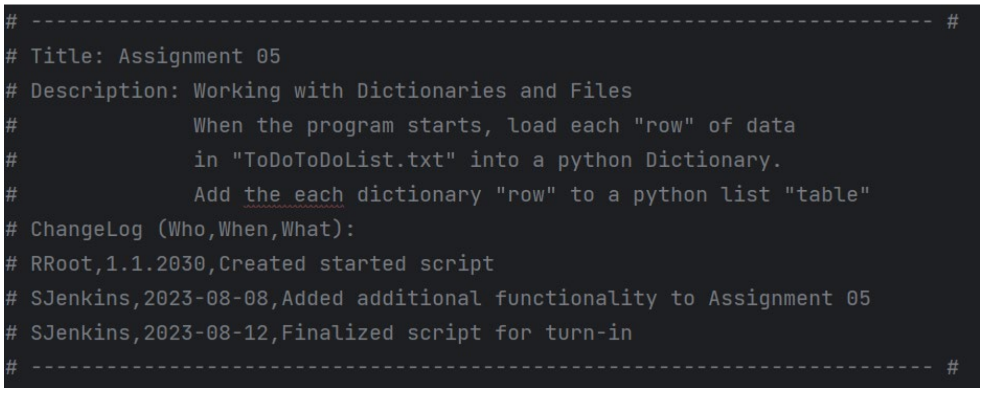
The new concepts that were utilized in this week's module have been covered with enough detail to get a foundational understanding of how this week's assignment was assembled. The next section will break down the actual code, as it was written, explain the layout, functions that were created and called, and how it was all assembled in the main body of the script.

This week's assignment required the students to work with a partially completed script, and to avoid use of any functions. For this reason the "Script Segments" section of the document has been removed and other than the header the "Main Body" will be covered.

The Header

The header will be added to each week's assignment as it is a best practice in programming to be detail oriented and provide as much information to those accessing the code as possible. This is a collection of fundamental information about the script, its creation and what it does.

- Information included with the Header
    - Title of the Script
    - A brief description of the functionality
    - A change log that includes the creation and any revisions to the script



```
# ---------------------------------------------------------------------- #
# Title: Assignment 05
# Description: Working with Dictionaries and Files
#              When the program starts, load each "row" of data
#              in "ToDoToDoList.txt" into a python Dictionary.
#              Add the each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# SJenkins,2023-08-08,Added additional functionality to Assignment 05
# SJenkins,2023-08-12,Finalized script for turn-in
# ---------------------------------------------------------------------- #
```

Figure 5: Header, as shipped with the "What is in Your House" script

Main Body

The main body is where the actual script originates, as Functions are stored into memory until they are called. At this juncture the script begins to fire off commands, starting the input sequence that will be interacted with by the user.

The segments of the script that are covered below are only the portions added to the original script, and does not cover every single line, including the options and exiting portions.

Variables – After "TODO: Add Code Here"

- `(Line 24) lstRow = []`
  - Variable to store a row of data from the associated text file.
- `(Line 25) strWarning = "Warning: Your To Do List Doesn't Exist!"`
  - Stored warning in case the script can't find an existing text document.
- `(Line 26) strTask = ""`
  - Empty variable used to store an entered task from the user.
- `(Line 27) strPriority = ""`
  - Empty variable used to store an entered priority for the previous task, from the user.

Processing – After "TODO: Add Code Here"

- `(Line 38) try:`
  - Set up an Error Handling stack, attempt the next set of statements.
- `(Line 39) objFile = open("ToDoList.txt", "r")`
  - Attempt to open the "ToDoList.txt" document and read it.
- `(Line 40) for row in objFile:`
  - If the file exists it will attempt to run a series of statements each row at a time.
- `(Line 41) lstRow = row.split(",")`
  - Returns the data into a list row.
- `(Line 42) dicRow = {"Task":lstRow[0].strip(), "Priority":lstRow[1].strip()}`
  - Stores each list row in a dictionary with the Keys of "Task" and "Priority".
- `(Line 43) lstTable.append(dicRow)`
  - Appends each Dictionary row to the main List Table.
- `(Line 44) objFile.close()`
  - Closes the opened document.
- `(Line 47) except:`
  - If the "Try" is unsuccessful will execute the following statements.
- `(Line 48) print(strWarning)`
  - Prints the stored warning, `"Warning: Your To Do List Doesn't Exist!"`
- `(Line 49) print("Creating One for You Now!\n")`
  - Let's the user know a new file will be created for them.
- `(Line 50) objFile = open("ToDoList.txt", "w")`
  - Create a text document named "ToDoList.txt", and write to the file.
- `(Line 51) objFile.close()`
  - Immediately close the document, readying it for the rest of the application.

- `(Line 52) print("Your To Do List is Created and Ready to Modify!")`
    - Let’s the user know that a file has been created.

Input/Output

Choice “1” - After “TODO: Add Code Here”

- `(Line 72) if lstTable == []:`
    - Checks if the List Table is empty.
- `(Line 73) print("Your To Do List is Empty, Try Adding Some Tasks!")`
    - If the Table is empty let’s the user know that they should add some tasks!
- `(Line 74) print("-" * 50)`
    - Prints a line-based divider.
- `(Line 75) else:`
    - If the Table is not empty execute the following statements.
- `(Line 76) print("Currently Your Tasks Are: \n")`
    - Header printed to show the user that what follows is their list of tasks.
- `(Line 77) for row in lstTable:`
    - Stores each row of the Table in a variable called “row”.
- `(Line 78) print("Task:", row["Task"], "||", "Priority:", row["Priority"])`
    - Prints each Task and Priority with formatting.
- `(Line 79) print("-" * 50)`
    - Prints a line-based divider.

Choice “2” - After “TODO: Add Code Here”

- `(Line 86) print("You've Chosen to Add a New Task!")`
    - Let’s the user know they have chosen to add a new task.
- `(Line 89) while(True):`
    - While loop checking if the user wants to exit the experience.
- `(Line 91) strTask = input("Type a New Task (or Exit for the Menu): ")`
    - Asks the user for a Task as input and stores it to the empty variable “strTask”.
- `(Line 92) if(strTask.lower() == "exit"):`
    - If statement to check if the user entered “exit”.
- `(Line 93) break`
    - If the user enters exit, break the loop.
- `(Line 96) strPriority = input("Type the Task's Estimated Priority[High/Medium/Low] (or Exit for the Menu): ")`
    - Asks the user for a Priority as input and stores it to the empty variable “strPriority”.
- `(Line 97) if(strPriority.lower() == "exit"):`
    - If statement to check if the user entered “exit”.
- `(Line 98) break`
    - If the user enters exit, break the loop.
- `(Line 101) else:`
    - If the user does not enter “exit” for either options proceed to the following statements.
- `(Line 102) print("You Entered:", strTask, "with a Priority of", strPriority)`
    - Print the enter values from the user, back to the user.

- `(Line 103) print("-" * 50)`
  - Prints a line-based divider.
- `(Line 104) lstTable.append({"Task":strTask, "Priority":strPriority})`
  - Append a dictionary entry to the Table.
- `(Line 105) print("You're Task has been Added! Remember to Save Before Exiting!\n")`
  - Let the user know their entry has been added and to remember to save before exiting.
- `(Line 108) strConData = str(input("Do you want to add more tasks? (y/n): "))`
  - Ask the user if they wish to enter more data before exiting back to the menu.
- `(Line 108) if(strConData.lower() == "y"):`
  - Check if the user entered “y” (yes).
- `(Line 110) print("\n")`
  - Add a new line for formatting reasons.
- `(Line 111) continue`
  - Proceed into the script.
- `(Line 112) elif(strConData.lower() == "n"):`
  - Check if the user entered “n” (no).
- `(Line 113) print("\n")`
  - Add a new line for formatting reasons.
- `(Line 114) break`
  - If the user opts to stop entering data, break the loop and go back to the menu.

Choice “3” - After “TODO: Add Code Here”

- `(Line 121) print("Follow the instructions below to remove a To Do List item!")`
  - Letting the user know they have chosen to remove items and to follow the prompts.
- `(Line 124) while(True):`
  - While loops that checks if the user wants to continuously remove data.
- `(Line 125) strRemoveTask = input("Which Task Would you Like to Remove?: ")`
  - Stores input from the user asking which Task they would live to remove from the list.
- `(Line 126) intRowCount = 0`
  - Establishing a counter variable.
- `(Line 127) blnRemove = False`
  - Establish a Boolean variable.
- `(Line 128) while(intRowCount < len(lstTable)):`
  - Loops while the counter is less than the length of the table (rows).
- `(Line 129) if(strRemoveTask == str(list(dict(lstTable[intRowCount]).values())[0])):`
  - Checks if the entered value is located in the saved list.
- `(Line 130) del lstTable[intRowCount]`
  - Delete the line if it is found.
- `(Line 131) blnRemove = True`
  - Sets the Boolean variable to True.
- `(Line 132) intRowCount += 1`
  - Increases the counter variable by one each loop.

- `(Line 135) if(blnRemove == True):`
    - Checks if the Boolean variable is true.
- `(Line 137) print("Your task was removed!")`
    - Let's the user know their selection was removed from the saved list.
- `(Line 138) strConRemoveData = str(input("Do you want to remove more tasks? (y/n): "))`
    - Asks the user if they would like to continue removing tasks.
- `(Line 139) if(strConRemoveData.lower() == "y"):`
    - Checks if the user entered a "y" (yes).
- `(Line 140) print("\n")`
    - Add a new line for formatting reasons.
- `(Line 141) continue`
    - Proceed into the script.
- `(Line 142) elif(strConRemoveData.lower() == "n"):`
    - Checks if the user enters a "n" (no).
- `(Line 143) print("\n")`
    - Add a new line for formatting reasons.
- `(Line 144) break`
    - If the user opts to stop removing data, break the loop and go back to the menu.
- `(Line 146) else:`
    - If the user enters an item not in the list, let the user know the task doesn't exist.
- `(Line 147) print("That task does not exist!")`
    - Print to the user that the task doesn't exist.

Choice "4" - After "TODO: Add Code Here"

- `(Line 155) strDispData = input("Do you wish to see your list before saving?: ")`
    - Asks the user if they would like to see the list of data before saving to the document.
- `(Line 156) if(strDispData.lower() == "y"):`
    - Checks if the user entered a "y" (yes).
- `(Line 157) print("Currently Your Tasks Are: \n")`
    - Header string to start the list to be displayed back to the user.
- `(Line 158) for row in lstTable:`
    - For each item in the table store it as the variable "row".
- `(Line 159) print("Task:", row["Task"], "||", "Priority:", row["Priority"])`
    - Print each Task and Priority row by row.
- `(Line 160) print("-" * 50)`
    - Prints a line-based divider.
- `(Line 163) objFile = open("ToDoList.txt", "w")`
    - Opens the "TodoList.txt" file, sets it to write, and stores it as objFile.
- `(Line 164) for row in lstTable:`
    - For each item in the table store it as the variable "row".
- `(Line 165) objFile.write("Task: " + row["Task"] + "||" + "Priority: " + row["Priority"] + "\n")`
    - Write each Task and Priority to the document row by row.

- `(Line 166) objFile.close()`
    - Closes the document after use.
- `(Line 167) input("Your tasks have been saved to your To Do List! Hit [Enter] to go back to the main menu.")`
    - Prompts the user to hit [Enter] to exit back to the main menu.

Summary

This week's assignment continued on the heels of the previous, asking for data, storing that data, and writing to an external file. Using concepts like Dictionaries the data was more efficiently stored and accessed, without having to know the index location. In a way this made the data more "human readable" and accessible.

The result is an intuitive application that asks the user to enter Tasks and associated Priorities, allowing them to see what they have entered, remove any entries they wish to remove, and save or write that data to an external text file. Utilizing new ways of organizing the functionality, through Try and Except the program can effectively correct any issues like missing files without causing confusing errors to the user.