Donavon Montgomery
May 15th 2023
Introduction to Python

Assignment 05- To Do List

Introduction

The assignment for this week was to add code to a " To Do List". Using dictionaries and files, I will be able to demonstrate how the list will be displayed, added, deleted, and saved.

Steps To Adding Code

The objective was to load each "row" of data in "TodoList.txt" into a Python dictionary and add each dictionary row into a Python list called "table". The next step was to show the current data, add a new item, remove an existing item, and save data to storage.The necessary inputs for successful execution included variables and constants defined at the beginning of the script such as the object representing a file, a row of text data from the file, a row of data separated into elements of a dictionary (Task, Priority), a list that acts as a 'table of rows, a menu of user options, and a variable to capture the user option selection.

Processing events started with Step 1, which loaded any previously-stored data from a text file called ToDoList.txt into a Python list of dictionary rows. The code responsible for making this happen was missing, but we will look at how we can add this in a later section.The Input/Output section started with Step 2, which displayed a menu of choices to the user. In this step, the user was presented with a menu with five distinct options to choose from. They included show current data, add a new item, remove an existing item, save data to file, and exit the program.Step 3 involved showing the current items in the ToDo list table. Here, the user was prompted with the option to view the existing tasks on the list.Next, Step 4 allowed users to add a new item to the list/table. The user was presented with a prompt to enter the task text followed by the priority weighted between low and high.The next step was to remove an existing item in the list/table. This involved users being prompted to indicate which task they would like to remove before proceeding.Lastly, in Step 6, the user was given the option of saving tasks to the ToDoFile.txt file. The prompt presented users with both the current items on the table and the option to either save the data or not. If Yes, each row of the data table was entered into the ToDoList.txt file, while the opposing option returns the user to the shown menu.Sample code involved the use of for loops to iterate through the codes, with output patterned using the python print function and user input captured through string logic and selection menus.

To make Step 1 function as intended, we have to add the code that reads the ToDoList.txt file and loads each row of data into a Python dictionary as shown below:

```
objFile = open("ToDoList.txt", "r")
```

```
for line in objFile:
    strData = line.strip().split(",")
    dicRow = {"Task": strData[0], "Priority": strData[1]}
    lstTable.append(dicRow)
objFile.close()
```

This reads the existing file, ToDoList, uses the python read-mode (“r”) function, and extends the application through the append function. The read file is then interpreted into dictionaries which are loaded into python lists for better organization.

Conclusion

In conclusion, all steps of this code were workflow-oriented with problems and goals clearly defined. Although some codes require additional input to be functional, understanding the logical processing that drives the code is an essential tool that programmers have to traverse unfamiliar scripts.