Task List

# Intro

In this assignment using a starter code written by someone else I wrote a script to record tasks and assign them priorities. To do this information from a txt file and user inputs were recorded into a list of dictionaries. Additionally, a menu was created to allow the user to interact with the list and perform operations on the list. The operations were to view what was in the list, add to the list, remove items from the list, save to file, and to close the program.

# Dictionaries

Dictionaries in Python are powerful data structures that allow us to store and retrieve data in key-value pairs. They are extremely versatile and widely used in Python programming. In a dictionary, each key is unique and associated with a corresponding value. The keys can be of various data types, such as strings, integers, or even tuples. Dictionaries are mutable, which means we can add, modify, or delete key-value pairs dynamically. Retrieving values from dictionaries is fast and efficient, as the lookup time is constant, regardless of the dictionary size. They provide a convenient way to organize and access data, making them invaluable for tasks like mapping, caching, and storing metadata. With their simplicity and flexibility, dictionaries are an essential tool in Python for handling and manipulating data1.

# Writing the script

This script began with a starter script that was provided to me. It contained three sections for data, processing, and for Input/Output. In data a set of variables were defined. Processing contained instructions to load data from a text file into a list of dictionaries. Input/Output contained the beginnings of a user menu and instructions for what the script should execute for a given option.

### Data

The data field consisted of variables that were defined for use throughout the script. These were mostly left the same apart from changing objFile to equal none and a new variable, strFile, that was set to the file name. This was done because the script ran into issues with objFile being set as a string. Another addition was made, to import os which will be discussed in the processing section (figure 1).

A screenshot of a computer program

Description automatically generated with low confidence

Figure 1. Data section of the script.

### Processing

To begin the script needed to take data from a text file and enter that into a list of dictionaries. This was done using a for loop and using ‘,’ as a separator, defining the first value on a line as a ‘task’ and the second value on a line as a ‘priority’. A couple problems arose with this method. The first if no file existed, it would return a FileNotFound error. To fix this a try-except statement was used were if there was a FileNotFound error, a file would be simply created. The other problem was if the text file empty, an empty dictionary was added to the list. To fix this os was imported and .getsize was used to check if the file was empty. If the was empty, it would be immediately closed and the creation of the list was done once user data was available (figure 2).

A screenshot of a computer screen

Description automatically generated with low confidence

Figure 2. Processing section of the script.

### Input/Output

To allow the user to interact with the script, a menu was used similar to previous assignments. The first option was to view current data, which when selected looped through the list and printed the values for the keys on each row. An if-elif statement was also used to print a message if the list had no data in it (figure 3).

A picture containing text, screenshot, font

Description automatically generated

Figure 3. Displaying data in the list to the user.

The second option allowed a user to add new items to the list. The script prompted the user to input a task and its priority and saved those as variables, which were then added to a dictionary and appended to the main list (figure 4).

A screen shot of a computer code

Description automatically generated with low confidence

Figure 4. Adding new items to the list as dictionaries.

The third option allowed the user to remove items from the list. This was done by asking user to input the task they wanted removed. The script then looped through the list and checked if items defined as ‘Task’ matched the user input. If they did, they were removed from the list (Figure 5). If time allowed a couple additions could be made to this option. One would be to check for errors in the user input such as if the item was not in the list or they used incorrect syntax. Another addition could be to allow the user to remove items by priority level. This could be done in a similar way fashion but checking the ‘Priority’ key instead. This would also require an additional input for the user to select remove by task or remove by priority.

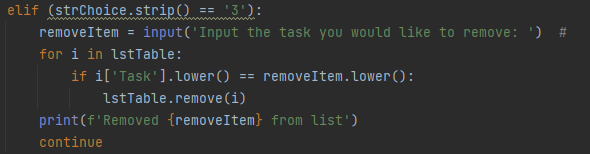


Figure 5. Removing an item from the list.

The fourth option allowed the user to save the data to the text file. This was done in the same way as the previous assignment but using dictionary keys instead of list items. The final option closed the program.

A screenshot of a computer

Description automatically generated

Figure 6. Code functioning within PyCharm.

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 7. Code functioning within the terminal.

# Summary

# Here, the use of a list of dictionaries provided a straightforward way to store and retrieve user data in a text file. By creating a dictionary for each task, information such as the task description and priority level could be easily organized and accessed. With the ability to add, modify, or delete key-value pairs dynamically, the flexibility and versatility of dictionaries proved essential for this task. By incorporating user input and a menu system, the script provided a user-friendly interface for managing the data in the text file1.

# Citations

1. OpenAI ChatGPT, May. 2023, chat.openai.com/chat: Aspects of this assignment were informed and created by queries I submitted to the ChatGPT.