Name: Joseph Géigel

Date: May 17, 2023

Course: Foundations of Programming: Python

Assignment: 05

GitHub URL: https://github.com/jgeigeluw/IntroToProg-Python

The To Do List Script: Working with Dictionaries

Introduction

The previous assignment paper discussed the use of lists and loops to create a script that produced an inventory of items in a house. This time we introduce the concept of *dictionaries*. With dictionaries, data is not stored in a sequence, but in pairs of *keys* and *values* that work similarly to a physical dictionary with words and definitions. Implementing dictionaries and while loops, the script discussed in this paper will allow users to review, add and remove tasks from a text file that contains a table of tasks and assigned priority values.

What is a Dictionary?

Unlike lists, dictionaries in Python are not sequences. Dictionaries let you organize information in pairs of *keys* and *values.* Like a physical dictionary, where each word has a definition, in a Python dictionary, each *key* gets a *value.* (Dawson, 2010) This introduces a fundamental difference between lists and dictionaries: accessing the elements within it. Elements in a list can be accessed by indexing, where an integer represents the position of a data element in the list. However, in dictionaries, data elements (values) are accessed by referencing their key. Keys are immutable, unique within a dictionary, and they are typically strings or numbers. (Dawson, 2010)

Creating Dictionaries

Another difference between dictionaries and lists can be found in the syntax of both data types. Values stored in a list are enclosed by brackets [ ] and separated by commas, while key-value pairs in a dictionary are enclosed by curly braces { }. Additionally, in dictionaries, key-value pairs are separated by commas, but the relationship between a key and its value is represented by a colon. (Dawson, 2010) This is an example of a dictionary that contains data about a student:

```
student = {"name": "Carlos", "age": 20, "major": "Geography",
                    "grades":[90,97,89,91]}
```

In this example, the name of the dictionary is `student` and it contains the name, age, major and grades of a student. The `name` and `major` keys contain string data values, `age` is an integer, and `grades` is a list of integers.

Accessing Dictionary Values

As mentioned earlier, keys are used to access the values in a dictionary, but there are a few ways of doing that. The most basic one is to retrieve the value by directly accessing it with a key, which can be done by simply entering the name of the dictionary and putting the key in brackets. (Dawson, 2010) In this example, the key `name` is used to retrieve the value `Carlos` from the `student` dictionary.

```
>>> student["name"]
'Carlos'
```

Another way of accessing the values in a dictionary consists in using the `get()` method. This method has the capability of handling situations where a key or value does not exist within in a dictionary, which would typically result in an error if the programmer simply tried to retrieve a value by entering a non-existent key. (Dawson, 2010) To use this method, you would enter the name of the dictionary, followed by `.get` and the desired key within parentheses. The example below shows how to use the get method to obtain the value `Carlos` from the `student` dictionary.

```
>>> print(student.get("name"))
'Carlos'
```

Adding are removing key-value pairs

Since dictionaries are mutable, you can add and remove key-pairs in them. Adding a key-value pair can be done by entering the name of the dictionary, followed by a key name in brackets, and an assignment operator that is followed by a new key value. (Dawson, 2010) Our `student` dictionary already showed the major of the student, but we can also store information about the student's minor. This example shows how to add that key-value pair.

```
student["minor"]= "Anthropology"
```

Removing a key-value pair from a dictionary is also possible. This can be done by using the `del` statement in Python. The delete statement must be followed by the dictionary name and the name of the key to be removed in square brackets. (Dawson, 2010) We can use this method to delete the key-value pair that we just added to our `student` dictionary.

```
del student["minor"]
```

The To Do List Script

This paper introduces a script that uses a dictionary to organize a series of tasks with different priority levels and writes that information to a text file. The script allows the user to perform several actions on the text file, such as: reviewing current data stored in the text file, adding new tasks and priorities, and removing tasks from the file.

Starter Script

One characteristic of this script is that it was created from an existing incomplete script that guided the development of the code. The starter code (figure 1) for the To Do List script included an outline of the general goals of the final product and possible steps to follow. The initial script suggested that the To Do List script needed to include: a section that loaded existing data to the script, a menu of options for the user, and five possible conditions that would guide the development of the code. These five conditions are determined by user input from the options displayed in the initial menu, and they allow these actions:

1. Show the current items in the table
2. Add a new item to the list/table
3. Remove an item from the list/table
4. Save tasks to the text file
5. Exit the program

```
# -- Input/Output -- #
# Step 2 - Display a menu of choices to the user
while (True):
    print("""
    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
    """)
    strChoice = str(input("Which option would you like to perform? [1 to 5] - "))
    print()  # adding a new line for looks
    # Step 3 - Show the current items in the table
    if (strChoice.strip() == '1'):
        # TODO: Add Code Here
        continue
    # Step 4 - Add a new item to the list/Table
    elif (strChoice.strip() == '2'):
        # TODO: Add Code Here
        continue
    # Step 5 - Remove a new item from the list/Table
    elif (strChoice.strip() == '3'):
        # TODO: Add Code Here
        continue
    # Step 6 - Save tasks to the ToDoToDoList.txt file
    elif (strChoice.strip() == '4'):
        # TODO: Add Code Here
        continue
    # Step 7 - Exit program
    elif (strChoice.strip() == '5'):
        # TODO: Add Code Here
        break  # and Exit the program
```

Figure 1. Portion of the starter script used to create the To Do List script.

Declaring variables

Four variables are declared at the beginning of the script (figure 2). The first one is the `objFile` variable that identifies the text file where a list of tasks and priorities will be written down. This file is stored in the same folder where the script is saved. After declaring that file object variable, the next one is `dicRow`, a variable representing a dictionary for each row of data where each *task* is a key, and each *priority* is a value. Then we have the lstTable variable, a list that acts as a table of data rows. Keys and values from the dictionary are appended to this list that will then be used to write to the text object file. Finally, we find `strChoice`, a variable that captures the menu option selected by the user of the program in order to perform one of the actions in the while loop.

```
13  # -- Data -- #
14  # declare variables and constants
15  objFile = "ToDoList.txt"   # An object that represents a file
16  dicRow = {}    # A row of data separated into elements of a dictionary {Task,Priority}
17  lstTable = []  # A list that acts as a 'table' of rows
18  strChoice = "" # A Capture the user option selection
```

Figure 2. Declaring variables and constants.

Loading existing data from file

This script populates a list of tasks and priorities by reading all the lines in an existing text file. To do this, the script opens a file object in read mode and iterates through each line using a for loop. In this for loop, a list of tasks and priorities is created by splitting each row at the comma that separates a task from its priority level. This results in a list for each line, where index value `[0]` represents the task name, and index value `[1]` represents the priority level assigned to that task. These two list items are used to define the values associated to the "Task" and "Priority" keys in the `dicRow` dictionary. A `dicRow` dictionary is created for each row of data, i.e. for each task, and then those key-pair (*Task-Priority*) values are appended to a list named `lstTable` by using the `.append` list method.

```
22  # -- Processing -- #
23  # Step 1 - When the program starts, load the any data you have
24  # in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)
25  objFile = open(objFile, 'r')                #Open file object in read mode
26  for row in objFile:                         #Iterate through lines in the text file
27      lstRow = row.split(",")                 #Create a list of "tasks" and "priorities" by splitting
28                                              # each line at the comma separator.
29      dicRow = {"Task": lstRow[0], "Priority": lstRow[1].strip()}     #Create a dictionary of tasks and priorities.
30      lstTable.append(dicRow)                 #Append dictionary key-value pairs to a list (table).
31  objFile.close()                             #Close file object.
32
```

Figure 3. Loading data from existing text file.

Displaying a menu of choices

Like in the previous *Home Inventory* script, the *To Do List* script presents the user a menu of choices that represents actions to be taken by the program based on the selection that the user makes. Each option is represented by a number that the user inputs when prompted by the program. These options include showing the current data, adding or removing tasks, saving the data to the text file, or exiting the program. Immediately after displaying the menu of options, the user is prompted to select an option by the question: *"Which option would you like to perform? [1 to 5]".* Then the user would select an option by entering one of the numbers in the menu.

```
33  # -- Input/Output -- #
34  # Step 2 - Display a menu of choices to the user
35  while (True):
36      print("""
37      Menu of Options
38      1) Show current data
39      2) Add a new item.
40      3) Remove an existing item.
41      4) Save Data to File
42      5) Exit Program
43      """)
44
45      strChoice = str(input("Which option would you like to perform? [1 to 5] - "))
46      print()  # adding a new line for looks
```

Figure 4. Display menu of choices.

Option 1: Show current tasks

If option #1 is selected by the user, the program will display the data that is currently in the `lstTable` list. This list is initially populated from data stored in the ToDoList.txt file object, but it could also contain tasks and priorities added by using the script that have not been yet written to the text file. Figure 5 shows the code implemented for option 1, where tasks and priorities in each row of `lstTable` are printed after a for loop iterates through all the lines in the list. The result of executing this option is seen in figure 6, where current data is presented in lines with tasks on the left, and priorities on the right.

```
48      # Step 3 - Show the current items in the table
49      if (strChoice.strip() == '1'):
50          print("Your current data is: \n")
51          for row in lstTable:              #Iterate through list
52              print (row["Task"] + ' | ' + row["Priority"])
53          input("\n" +"Press enter to continue.")
54          continue
```

Figure 5. Display current data in list.

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

Your current data is:

read a book chapter | high
eat | high
sleep | high
play Civilization | low

Press enter to continue.
```

Figure 6. Output from selecting option 1.

Option 2: Add new tasks and priorities

When option #2 is selected, the program allows the user to add new tasks and priorities to the list (`lstTable`). Tasks added here are not immediately saved to the text file, but they can be saved by selecting option 4 after adding the tasks, or by saving to file before exiting the program (option 5). Figure 7 shows a *while* loop within the *elif* statement that executed option 2. In this while loop, two user inputs are captured: `strTask` and `strPriority`. These two inputs are used to populate the new key-value pair that is appended to `lstTable` in line 60 of the script (see figure 7). After the newly added tasks and priorities are appended to the list, the `strChoice` input variable is presented again, but this time to ask the user if more tasks need to be added. The reason why a while loop was implemented here is to allow the user to add additional tasks after the first one if desired. This is accomplished by the `continue` statement that runs if the user selects “y” after the “Add more tasks?” prompt.

```
54      # Step 4 - Add a new item to the list/Table
55      elif (strChoice.strip() == '2'):
56          while(True):
57              print("Type in a task and assign a priority level: ")
58              strTask = input("Task: ")
59              strPriority = input("Priority: ")
60              lstTable.append({"Task":strTask,"Priority":strPriority})
61              for row in lstTable:
62                  print (row["Task"] + ' | ' + row["Priority"])
63              print()
64              strChoice = input("Add more tasks? ('y/n'): ")
65              if strChoice.lower()== 'y':
66                  continue
67              elif strChoice.lower()=='n':
68                  break
```

Figure 7. Adding new tasks.

Option 3: Removing a task

Option #3 allows the user to remove a task from the list. This option captures the name of the task to be removed by using the `strTask` input variable. The program then implements a for loop to find the task to be removed, and the `remove` list method to take the row associated to the selected task off the list.

```
70      # Step 5 - Remove an item from the list/Table
71      elif (strChoice.strip() == '3'):
72          strTask = input("Task to Remove: ")
73          for row in lstTable:
74              if row["Task"].lower()==strTask.lower():
75                  lstTable.remove(row)
76                  print("Task removed.")
77              else:
78                  print("Task not found.")
79          continue
```

Figure 8. Removing tasks from list.

Option 4 and 5: Saving data to the text file and exiting the program

Finally, the user can select from options 4 and 5 after reviewing, adding or removing tasks. Option #4 lets the user save current list data to the ToDoList.txt file. During the execution of the program, changes made to the list of tasks and priorities are only stored in memory, and not automatically saved to the file object. If option #4 is selected, the programs opens the ToDoList.txt file object in write mode and uses the `write` method to write each row in `lstTable` to new lines in the text file. After writing down all items in the list (table), the program closes the file. The user can then select another option to make more changes to the list of tasks or to exit the program.

The last alternative is option #5. This option allows the user to exit the program. When selecting this option, the user is prompted with the question *"Would you like to save your data?"*. If the user selects to save the data, the program performs the same actions that would be executed by selecting option #4: opening file in write mode, writing down tasks, and closing the file, and then exits the program. This allows the user to save the data and close the program in only one option selection or if option #5 was selected by error. If the user chooses not to save the data, the program simply closes and no changes are saved. Figure 9 shows the Python code associated to saving the data and exiting the program.

```
    # Step 6 - Save tasks to the ToDoList.txt file
    elif (strChoice.strip() == '4'):
        objFile =  open("ToDoList.txt","w")     #Open file object in write mod
        for row in lstTable:                    #Iterate through list of tasks
            objFile.write(str(row["Task"])+','+str(row["Priority"]+'\n'))   #W
        objFile.close()                         #Close file object.
        print("Data added to file!")
        continue

    # Step 7 - Exit program
    elif (strChoice.strip() == '5'):
        print("Would you like to save your data?")      #Prompt the user to ch
        sav = input("Enter 'y' or 'n': ")   #User chooses to save or not to sa
        if sav == "y":
            objFile = open("ToDoList.txt", "w")         #Open file object in w
            for row in lstTable:                        #Iterate through list
                objFile.write(str(row["Task"])+','+str(row["Priority"]+'\n'))
            objFile.close()                             #Close file object.
            print("Data saved!")
            break
        elif sav == "n":                #Exit program if user chooses not to s
            break
        else:
            print("Please enter 'y' or 'n'")
            continue
        break  # and Exit the program
```

Figure 9. Saving data and exiting.

Summary

Previously we had examined how to store data in sequences, such as lists, but this paper introduced an alternative and powerful way to organize data by using dictionaries. Dictionaries store information in key-value pairs where each key represents a value in the same way that a word represents a definition in a physical dictionary. In addition, this paper discussed how to create, access and modify dictionaries. Finally, we implemented dictionaries to develop a To Do List script that allows the user to review, add or remove tasks and priorities from a text file object using Python.

References

Dawson, M. (2010). *Python Programming for the Absolute Beginner* (Third Edition ed.). Boston, MA, United States of America: Course Technology PTR.