Elena McDonald

May 12, 2021

Foundations of Programming: Python

Assignment 05 – Lists and Dictionaries

# Module 5: Lists and Dictionaries

## Introduction

Learning objective of module 05 is to create scripts using Lists and Dictionaries, get started with GitHub and learn best practices for organizing code (separation of concerns and scrip templates).

## Lists

Lists are ‘mutable’ as opposed to strings and tuples. They can be added to (appended). List can hold a variety of data types and are defined by square brackets. Lists can be collections of tuples – in this case lists are like tables that hold rows of data (strings, tuples, dictionaries). These rows inside a list are comma-separated.

## Dictionaries

Dictionaries are collections of data that store information in pairs. Each pair of data has ‘key’ and ‘value’ element. It is like conventional dictionaries that list words and word definitions. But in Python dictionaries can hold any type of ‘keys’, for example:

{“name”:”John”,”phone”:”(206)230-1234”,”registered”:”yes”}

Dictionaries can be represented as tables, where ‘key’ is the column name and “value” is cell value. If dictionaries are grouped in a list [ ] they represent rows in a table.

## Loading list data from a file

Data stored in a text file has to undergo conversion and be loaded back to memory so the program can perform operations with it. Figure 1 shows how to load list data from a simple text file:

objFile = open(**"ToDoList.txt"**, **"r"**)  
for row in objFile: *# EM: loops through file and loads data to list* strData = row.split(**","**)

*Fig. 1: Example of loading list data from text file. Data is comma-separated.*

If text file contains rows of comma-separated values, rows can be loaded as dictionaries and grouped in a list, as shown in Figure 2 below.

objFile = open(**"ToDoList.txt"**, **"r"**)  
for row in objFile: *# EM: loops through file and loads data to list* strData = row.split(**","**)  
 dicRow = {**"task"**: strData[0],**"priority"**: strData[1].strip()}  
 lstTable.append(dicRow)  
objFile.close()

*Fig. 2: Example of loading list data from text file. Data is comma-separated.*

Here each row is first split on comma and loaded into memory as a dictionary {curly brackets}. Then each dictionary row is added (appended) to the list (Table of data)

## Writing data to a file

Data stored in memory as lists and dictionaries can be loaded to text file with some transformation. Data can be loaded as strings, separated with commas and new line sign to break data into rows inside the text file. Figure 3 illustrates how this can be accomplished for list data – it uses [subscript] to identify data:

objFile.write(lstRow[0] + **','** + lstRow[1] + **','** + lstRow[2] + **'**\n**'**)

*Figure 3. Example of writing list data to a text file.*

Data can be loaded to a file row by row, when loops are used. This is illustrated in Figure 4, where dictionary data stored as a list is loaded into text file:

objFile = open(**"ToDoToDoList.txt"**,**"w"**) *# EM: opens new file and writes all tasks to it*for row in lstTable:  
 objFile.write(str(row[**'task'**]) +**','** + str(row[**'priority'**]) + **'**\n**'**)  
objFile.close()

*Figure 4. Example of writing list of dictionaries into a text file.*

## Removing data from a list

Data can be removed from a list by using remove function, where row represents index of a row in a ‘for’ loop:

lstTable.remove(row)

## Script Templates

Can be used to organize code and increase consistency and readability. Any Python file can be stored as a ‘starting’ script template.

It is a good practice to organize code into 3 main sections: “Data”, “Processing” and “Input and Outpur” (or “Presentation). These main sections can be included in the template as a starting point and a reference.

## Assignment 05

Objective: To modify a script that manages a "ToDo list." The "ToDo" file will contain two columns of data, "Task" and "Priority." Load the columns into a Python Dictionary object. Each dictionary object represents one row of data, and these rows must be added to a Python *List* object to create a table of data. Modify “TO DO” sections in the provided script.

### Step 0

I downloaded starting script "Assigment05\_Starter.py" to Assignment05 project in PyCharm. I also created a starting text file ToDoList.txt and loaded 2 rows of data into it, representing a task and its priority (1-5]:

make bed,1  
floss,2

### Step 1

For step 1 I added code that loads data from my text file into Table. Code loops through the rows of data, loading them first as dictionary rows and then adding rows to the List (Table):

*# Step 1 - When the program starts, load any data you have  
# in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)*objFile = open(**"ToDoList.txt"**, **"r"**)  
for row in objFile: *# EM: loops through file and loads data to list* strData = row.split(**","**)  
 dicRow = {**"task"**: strData[0],**"priority"**: strData[1].strip()}  
 lstTable.append(dicRow)  
objFile.close()

### Step 2

In TO SECTION for Step#3 I added code for printing List data to a screen:

for objRow in lstTable: *#EM: prints every dicRow in lstTable* print(objRow)

### Step 3

Next I added script for adding new tasks to a list. First, user input is collected and converted to a dictionary row, then added (appended) to the list:

while (True):  
 strNewTask = str(input(**"Please enter task: "**)) *# EM: collect input for new task* strNewPriority = str(input(**"Please enter priority 1 to 5: "**))  
 dicRow = {**"task"**: strNewTask.strip(), **"priority"**: strNewPriority.strip()}  
 lstTable.append(dicRow) *# EM: add new row to the list/Table* strChoice = input(**"Exit? (y/n): "**)  
 if strChoice.lower() == **"y"**:  
 print(lstTable) *# EM: prints list with new rows added* break

### Step 4

Step 5 in the program requested code for removing an item for the list/table. After user enters which task is to be removed, the task is compared to values with key ‘task’ – if there is a match in the list, the entire row (including task’s priority) is getting removed. I struggled with this part of assignment. In the end program worked when I added .strip to my IF condition:

strItem = str(input(**"Task to remove: "**))  
for row in lstTable:  
 if row[**"task"**].lower().strip() == strItem.lower().strip():  
 lstTable.remove(row)  
 print(**"Task removed!"**)  
 else:  
 print(**"Task does not exist. Please try again."**)

### Step 5

Step 6 requested to write list/table data to a new file, “ToDoToDoList.txt”. Each row of data is written to the file as concatenated string of key values, comma, and a carriage return at the end of each row, to separate rows in the file when stored. This is done via FOR loop, which iterates through dicRows one by one until all rows in the List are written to text file:

objFile = open(**"ToDoToDoList.txt"**,**"w"**) *# EM: opens new file and writes all tasks to it*for row in lstTable:  
 objFile.write(str(row[**'task'**]) +**','** + str(row[**'priority'**]) + **'**\n**'**)  
objFile.close()  
print(**"File updated!"**)

### GitHub

I uploaded my work to GitHub repository as directed.

### Testing in PyCharm

### Data loaded from the file, and printed to the screen when option 1 selected:

Text

Description automatically generated

New tasks added when option 2 is selected:

Text

Description automatically generated

The task ‘repeat’ was removed from the list:

A picture containing chart

Description automatically generated

When menu option 4 was selected, all tasks were written into new text file:

![Graphical user interface, text, application

Description automatically generated]()

### Testing in Command Line Mac OS:

In Mac OS my code only worked when I provided absolute path for the text file in my code:

Graphical user interface, text, application

Description automatically generated

I added 3 more tasks:

Text

Description automatically generated

and removed sleep:

Text

Description automatically generated

and wrote data to the file:

Graphical user interface, application

Description automatically generated