MAPÚA UNIVERSITY
SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 2:
Strings, Lists, Tuples, and Dictionaries

CPE106L (Software Design Laboratory)

Darren M. Briones Date: 2021. 03. 19 23:26:15
Edmund Hans Tapado Date: 2021. 03. 19 18:06:15
Justine Angelo Ariel V. Villespin Date: 2021. 03. 19 21:32:03

Group No.: **Group 3**
Section: **B2**

PreLab

Readings, Insights, and Reflection

Insights

This chapter tackles three types of containers and the methods used within the containers. Namely, lists, tuples, and dictionaries (all of these can contain data of any types). Lists are mutable (which essentially means it can be modified throughout the usage of the program) through the use of "index" and assigning values you see fit. In contrast, tuples is just like a list, but immutable. Which means the values within the container are already defined in the code, without any interference from the user (it is set). Lastly, dictionaries are containers that organized information by association. To organize information, a key and a value are paired. **(BRIONES)**

In the chapter, we mainly discussed the containers used for different types of variables, these containers are called list, dictionaries, and tuples. These containers can be accessed through their index and be manipulated. The difference between a list and a tuple is that a list is mutable, and a tuple is immutable. While a dictionary has a key-value pair. **(TAPADO)**

While performing the various tasks in the Laboratory report 2, I realized that there is a difference between the use of list and tuple. The data type list in mutable while tuple is immutable, which means that the value of a tuple data type cannot be changed. **(VILLESPIN)**

Reflection

Before reading this chapter, I already had knowledge about data containers because of my past experience coding with the language C++, in C++, I have used arrays (1 dimensional and 2 dimensional) and vectors (which in my opinion, is a superior version of arrays). Although I've known about these types, I had a hard time putting my intentions into code, there were a lot of restriction (data types, etc...) that left a bad taste in my mouth. However, learning about lists, tuples, and dictionaries, it became clear to me that python was the way to go. It was suddenly so much easier to handle data. **(BRIONES)**

A program can be created in different languages such as, C++, C#, Python, Java, etc., however, the algorithm behind a program can be done similarly or in different ways. After studying the chapter related with the lab report, I now understand that the basics of programming is simply how a person formulate the right logical statements for the program to run properly. In addition, a container is very essential when you want a specific group of datatypes or information to be stored together because it allows easy accessibility. **(TAPADO)**

In this Lab report 2, I was given a chance to learn and discover a new topic for python programming. Also, I learned that objects that are immutable are quicker to access than mutable objects. While doing the laboratory report, I realized that python programming is a good programming language but difficult to use. However, if I practice it, I think I can manage to create programs using python programming language. **(VILLESPIN)**

Answers to Questions

1. B (20)
2. B ([20, 30])
3. A (1)
4. B ([10, 20, 30, 40, 50])
5. B ([10, 5, 30])
6. C ([10, 15, 20, 30])
7. B ([“name”, “age”]
8. B (None)
9. B (pop)
10. B (strings and tuples)

InLab

- **Objectives**
  1. **Use** Linux terminal in running python statements.
  2. **Use** Visual Studio Code of Linux in debugging programs.
- **Tools Used**
  - Anaconda
  - Linux Terminal
- **Procedure.**

In Figure 1.1, we used Visual Studio Code of Linux to debug and run the program. We activated first kivy for the python program to run. Without kivy, we cannot run the python program. Next, we debug and run the first program. After compiling and running, the program is asking for a number. We typed in 32 to be the number that we want.

Figure 1.1. Start of the program while activating kivy to run python programs and debugging computesquare.py program.

In Figure 1.2, after typing in 32, the square of 32 was shown since the program is focused in getting the square of the input number in the program. and This ends the computesquare.py program.

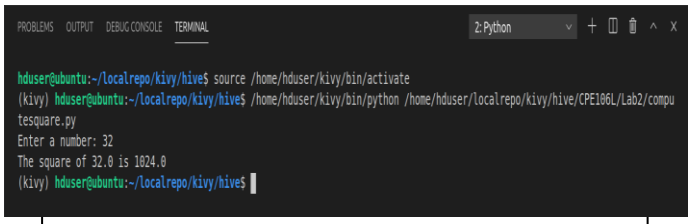

Figure 1.2. Output and end of the program.

In Figure 1.3, we tested a number with a decimal point to be our desired number to be typed in. We chose 22.52 as our number and the square of 22.52 is 507.1504. Inputting numbers with decimal point is possible since the data type that was used for this input is float.

Figure 1.3. Testing a number with a decimal point

In Figure 1.4, we run computesquare.py in Linux terminal. We typed in 12.113 to be the number. After that, we got an output of 146.72476899999998. Moreover, we can see the output difference when we debug and run it in Visual studio code and running it in Linux Terminal.

Figure 1.4. Running computesquare.py in Linux Terminal.

In Figure 2.1, we can see that we are debugging and running doctor.py program. When we debug and run it, an output was shown. The program output its greeting and in the next line, it asks for an input on what the program can do for us. We input checkup.

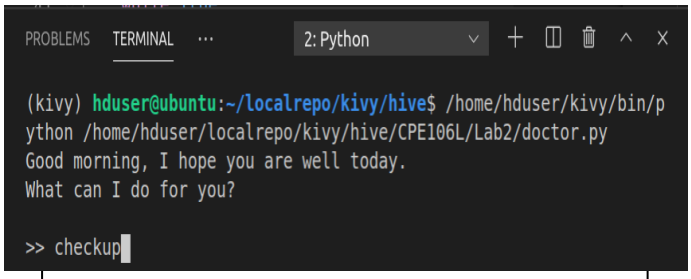

Figure 2.1. Start of the program.

In Figure 2.2, we can see that a new output was shown asking us an explanation for why we are seeking for a checkup. I input I have a high fever as our input in this step.

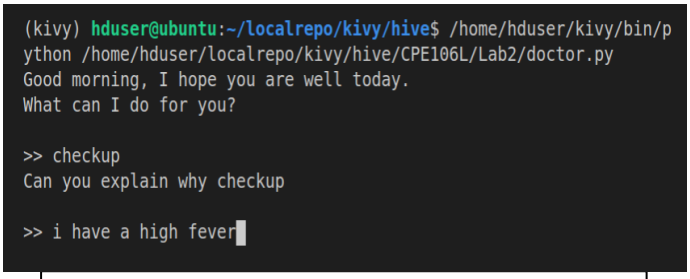

Figure 2.2. The program asking a new question.

In Figure 2.3, we wanted to end the program. For us to end the program, we need to type 'QUIT' for us to end the program and after typing it, we get a final output of "have a nice day!".

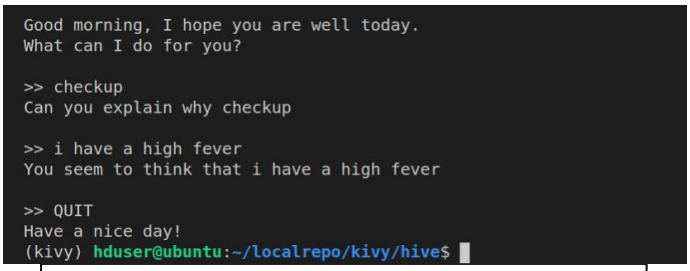

Figure 2.3. Ending the program

In Figure 3.1, the program asks us the number of sentences we want to be inputted. I input 6 to be the number of sentences.

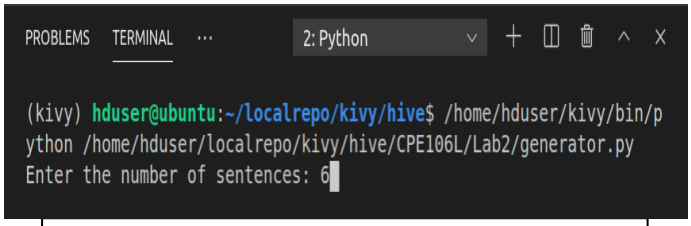

Figure 3.1. Start of the program

In Figure 3.2, we can see the words that would be used for us to see the output random generated sentences. Articles, nouns, verbs, and prepositions are tuples that would be used in our program.

```
articles = ("A", "THE")

nouns = ("BOY", "GIRL", "BAT", "BALL")

verbs = ("HIT", "SAW", "LIKED")

prepositions = ("WITH", "BY")
```

Figure 3.2. The words that would be used to be a sentence.

In Figure 3.3, we input 6 sentences and we got these 6 random constructed sentences. This was the output result since this program was a random generator of sentences with a pool of words that is put together to construct a sentence. In Figure 3.2, we can see the words that would be constructed as a sentence.

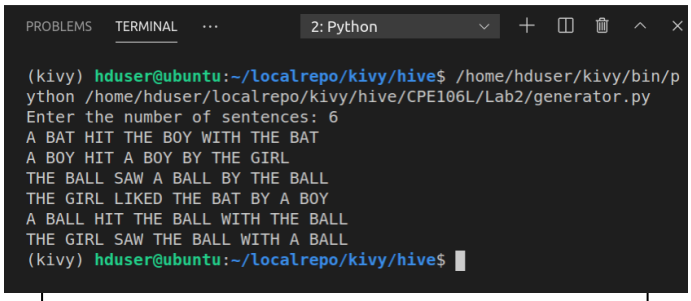

Figure 3.3. The output of the program and end of the program.

In Figure 3.4, we run the program in Linux terminal. We input 10 to be the number of sentences to be shown. After inputting 10, we got 10 random generated sentences.

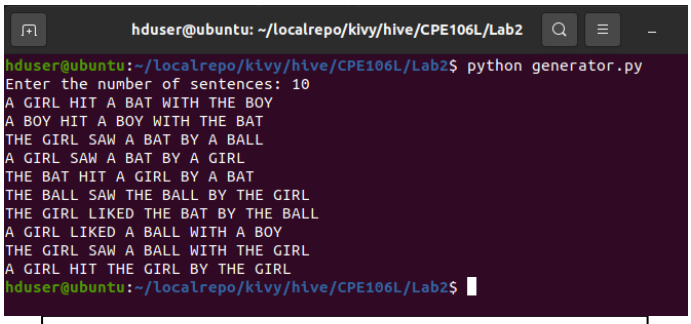

Figure 3.4. Running generator.py in Linux terminal.

In Figure 4.1, we can see that binary numbers were only shown as an output for the program.

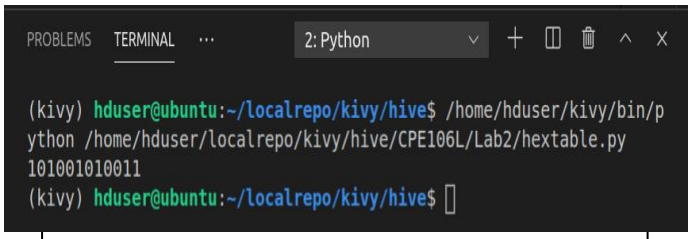

Figure 4.1. Start and end of the program.

In Figure 4.2, 35A is the hexadecimal value that will be converted to a binary value. This program operates as a converter of hexadecimal values to binary values. The binary value for 35A as shown in Fig 4.1 is 101001010011.

```
print(convert("35A", hexToBinaryTable))
```

Figure 4.2. The hexadecimal value that will be converted to binary.

In Figure 4.3, we run hextable.py in Linux terminal. We tried a new hexadecimal value which is EA and we got an output of 10101110.

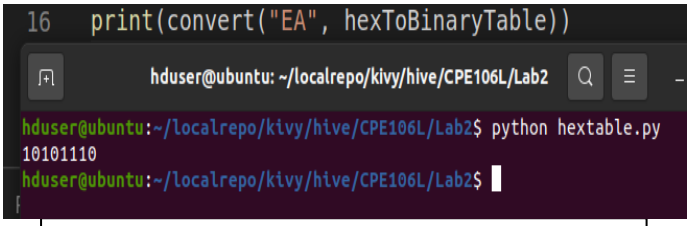

Figure 4.3. Running hextable.py in Linux terminal.

In Figure 4.4, we tested a small letter 'a' to be converted to a binary value. However, the program showed an error since 'a' is not in the hexadecimal table since hexadecimal starts from 0-9 and A-F.

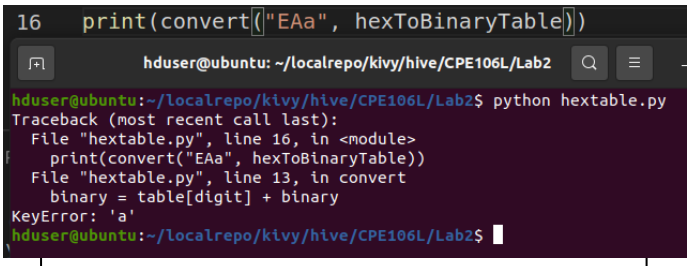

Figure 4.4. Testing a small letter to be converted.

In Figure 5.1, the program asks us to enter a file name that will be used for the program. I input numbers.txt as the file name to be used in the program.

Figure 5.1. Start of the program.

In Figure 5.2, It shows the numbers that is contained inside numbers.txt file. This .txt file will be used for the program.

Figure 5.2. Numbers inside numbers.txt

In Figure 5.3, after typing in numbers.txt, we got the median output which is 31.0. the median was calculated using the numbers inside numbers.txt file.

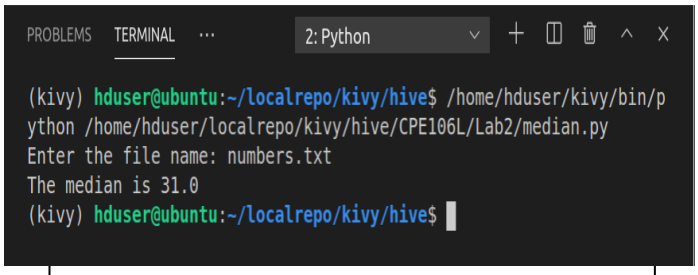

Figure 5.3. End of the program

In Figure 5.4, we run median.py in the Linux terminal. However, when we run it in the Linux terminal, there was an error that occurred during the process. So, we were not able to run it here in Linux terminal, but it can be run in Visual Studio Code terminal.

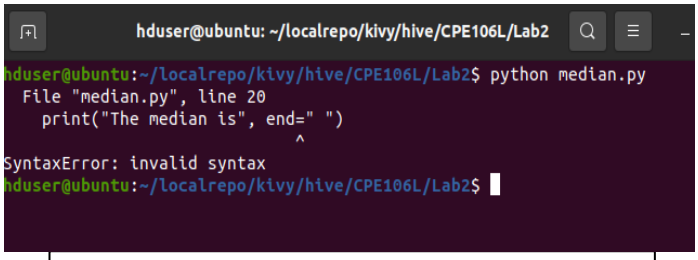

Figure 5.4. Running median.py in Linux terminal.

In Figure 6.1, we have the same step in Fig 5.1 that will be done here. The program asks us to enter a file name that will be used in the program.

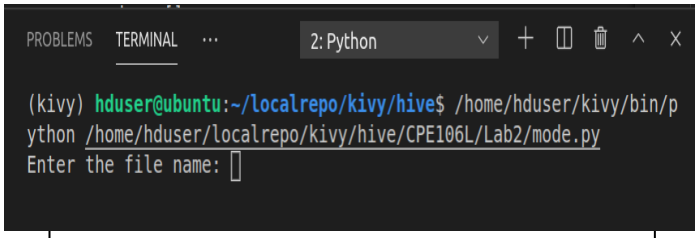

Figure 6.1. Start of the program.

In Figure 6.2, we used the same file name which is numbers.txt, but this program is different. This program is made to solve for the mode. The mode in numbers.txt file is 15.

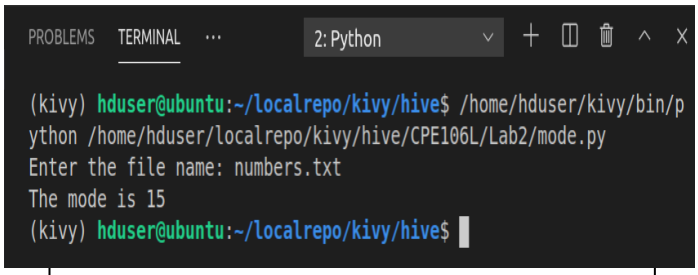

Figure 6.2. End of the program.

In Figure 6.3, we run mode.py in the Linux terminal. When we input the file name, an error has occurred in the process. So, we were not able to run it here in Linux terminal, but it can be run in Visual Studio Code terminal.

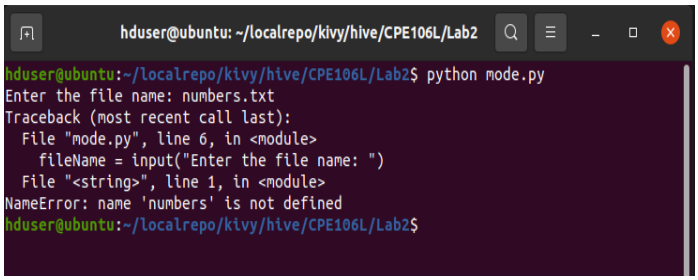

Figure 6.3. Running mode.py in Linux terminal.

PostLab

Programming Problems

In Fig 7, the program prompts the user to input an existing .txt file and checks whether the file is empty or not. If the file is empty, the program prints an "Empty file" message and terminates the program. If the file contains a value or values the user may either choose to find its mean, median, or mode.

Figure 7. Running Stats.py program.

In Fig 8, the program counts the number of lines that exists in a .txt file, then, the user may enter a line number and the program returns the content of that line number.

Figure 8. Running LR2_2.py program.

In Fig 9, the program defines each .txt file at the start of the program using the getWords() function. The user then may input the number of times the program will generate a randomized sentence.

Figure 9. Running Generator_modified.py program.