MAPÚA UNIVERSITY

SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 2: Strings, Lists, Tuples, and Dictionaries

CPE106L-4 (Software Design Laboratory)

Member 1: Aljehro R. Abante
Member 2: Cerdan Karl T. Alarilla
Member 3: Daryl Jake A. Fernandez

Group No.: **3**
Section: **E01**

PreLab

Readings, Insights, and Reflection

Fundamentals of Python: First Programs
Kenneth A. Lambert
Edition 3
ISBN: 9780357881132

Insights and Reflections

Abante (Chapter 4: Strings and Text Files)

- In Chapter 4, we learned about strings and text files, which is the centerpiece of computing and text editing. The chapter explains how a user can change their strings, which explains why they can access and index the strings' characters, making them vital for purposes such as bringing out data values and even coding them. The length of strings and their position are also other things that work well to allow the programmer to work with the text quickly and appropriately. In the same manner, the chapter provides the basic string techniques and file handling procedures and explains how to create, open, read, and write different text files which are the basic practices for any programmer. All these principles are included in a wide range of applications such as programming of websites and drawings of borders around or including images, and even data mining, where these two can transform and manage data textually within the given scope. In addition, reviewing these tenets, one appreciates the fact that skills in string handling and file management, are critical in programming with Python as well when dealing with data.

Alarilla (Chapter 5: Lists and Dictionaries)

- The chapter 5 of the book explains the data structures of lists and dictionaries. Lists and dictionaries are useful tools for organizing and manipulating data effectively. Similar to how strings work, they are both sequences where elements are indexed according to their position which allows access and manipulation of items. They can store any data type, from numbers to strings to even nested lists, making them versatile for various applications like creating rosters or storing ordered data. Dictionaries on the other hand provides non-sequential structure. It means that dictionaries allow quick lookups by key rather than by position making it more ideal for data association like storing attributes. Unlike lists, dictionaries are unordered and focuses more on key-based access.

Answers to Questions

For Questions 1-6, assume that the variable data refers to the list [10, 20, 30].

1. The expression data[1] evaluates to 20. **(b)**
2. The expression data[1:3] evaluates to [20, 30]. **(b)**
3. The expression data.index(20) evaluates to 1. **(a)**
4. The expression data + [40, 50] evaluates to [10, 20, 30, 40, 50]. **(b)**
5. After the statement data[1] = 5, data evaluates to [10, 5, 30]. **(b)**
6. After the statement data.insert(1, 15), the original data evaluates to [10, 5, 30]. **(b)**

For questions 7–9, assume that the variable info refers to the dictionary

7. The expression list(info.keys()) evaluates to ["name", "age"]. **(b)**
8. The expression info.get("hobbies", None) evaluates to None. **(b)**
9. The method to remove an entry from a dictionary is named pop. **(b)**
10. Which of the following are immutable data structures? **b. string and tuples**

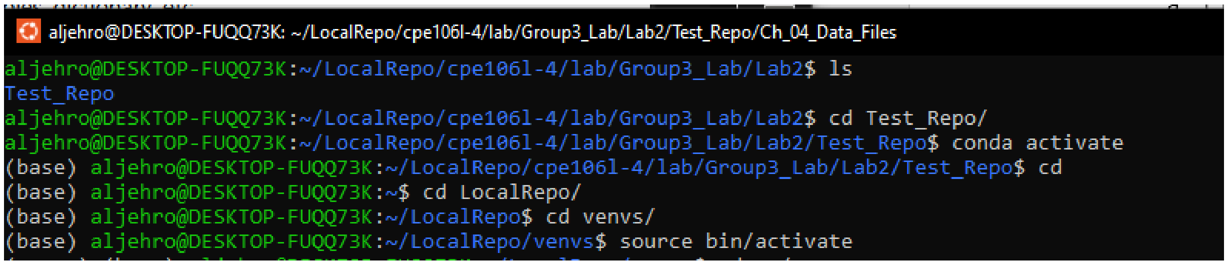
InLab

- **Objectives**
  1. **Debug** the sample programs on tuples, dictionary, etc.
  2. **Use** Anaconda (or Python Virtual Environment) and Linux terminal in running python statements.
  3. **Use** Visual Studio Code in debugging.
  4. **Compare (Discuss the difference)** C++ and Python data structures.

- **Tools Used**
  - Anaconda (or Python Virtual Environment)
  - Ubuntu Linux Virtual Machine
  - Visual Studio Code

- **Procedure**.

1. The first thing that we have to do is to first use wsl to interact with Ubuntu. This will serve as a machine to test the codes. Then proceed to activate Python virtual environment or Anaconda.



```
aljehro@DESKTOP-FUQQ73K: ~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files
aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2$ ls
Test_Repo
aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2$ cd Test_Repo/
aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo$ conda activate
(base) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo$ cd
(base) aljehro@DESKTOP-FUQQ73K:~$ cd LocalRepo/
(base) aljehro@DESKTOP-FUQQ73K:~/LocalRepo$ cd venvs/
(base) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/venvs$ source bin/activate
```

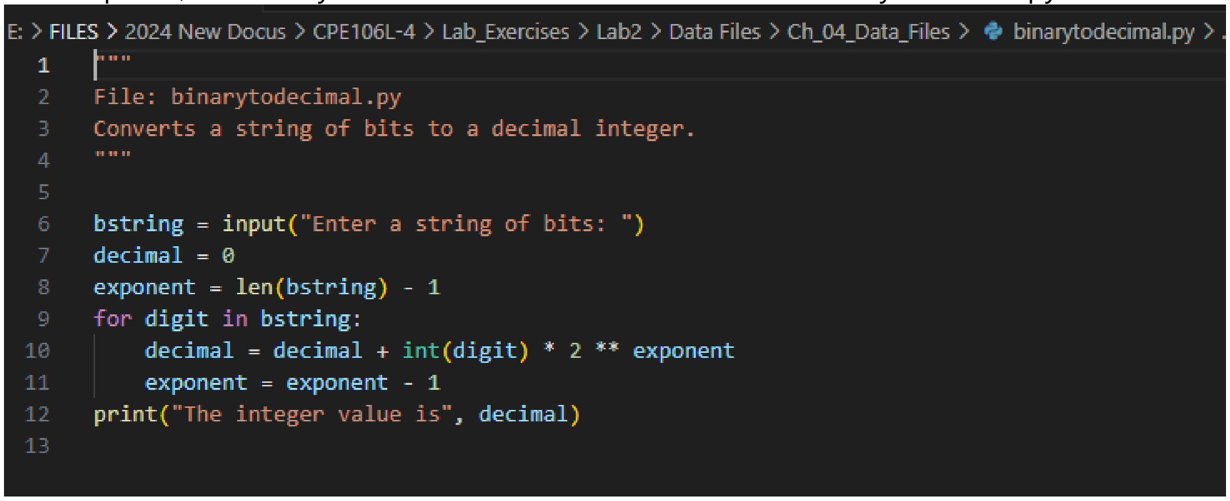
Figure 1

- As seen in figure 1, this is how we activated wsl Ubuntu and proceed on activating both Python virtual environment, and Anaconda. In my case this case, both were activated to show how it is activated.

2. Then now we open VS Code to proceed on debugging the python files.

Chapter 4

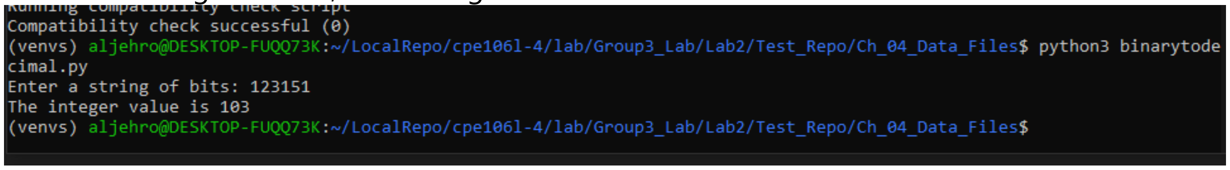
For Chapter 4, the first Python file that we have to check is the binarytodecimal.py:



```
E: > FILES > 2024 New Docus > CPE106L-4 > Lab_Exercises > Lab2 > Data Files > Ch_04_Data_Files > binarytodecimal.py >
"""
File: binarytodecimal.py
Converts a string of bits to a decimal integer.
"""

bstring = input("Enter a string of bits: ")
decimal = 0
exponent = len(bstring) - 1
for digit in bstring:
    decimal = decimal + int(digit) * 2 ** exponent
    exponent = exponent - 1
print("The integer value is", decimal)
```

Figure 3

- After checking the code, run through the terminal:

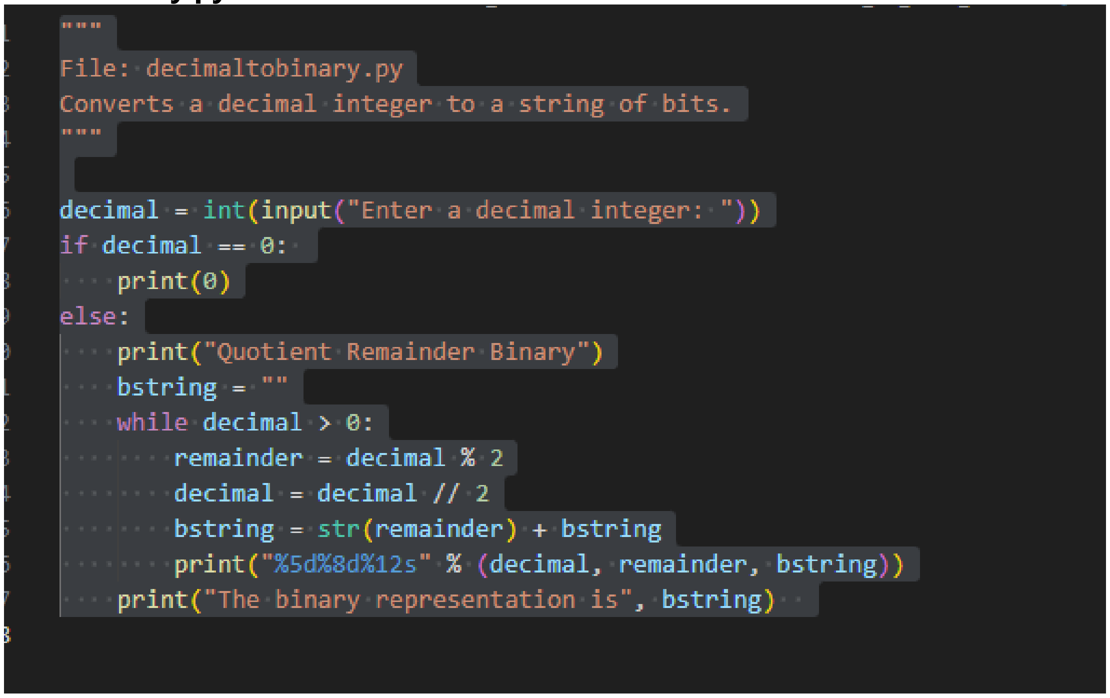


```
Running compatibility check script
Compatibility check successful (0)
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$ python3 binarytode
cimal.py
Enter a string of bits: 123151
The integer value is 103
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$
```

Figure 3.1

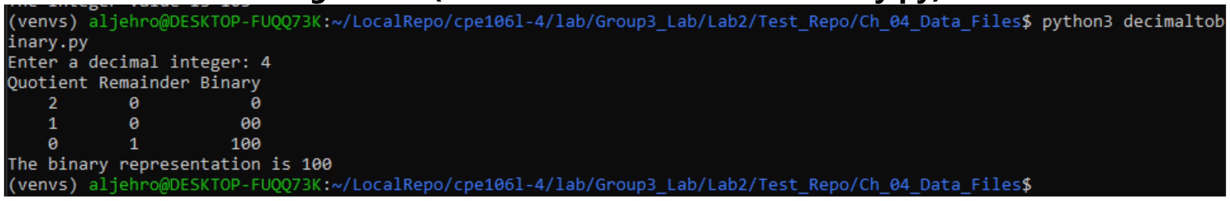
-As we can see, the program has no problems in debugging, we now proceed on debugging the provided Python Files and the rest of Chapter 4

#decimaltobinary.py



```python
"""
File: decimaltobinary.py
Converts a decimal integer to a string of bits.
"""

decimal = int(input("Enter a decimal integer: "))
if decimal == 0:
    print(0)
else:
    print("Quotient Remainder Binary")
    bstring = ""
    while decimal > 0:
        remainder = decimal % 2
        decimal = decimal // 2
        bstring = str(remainder) + bstring
        print("%5d%8d%12s" % (decimal, remainder, bstring))
    print("The binary representation is", bstring)
```

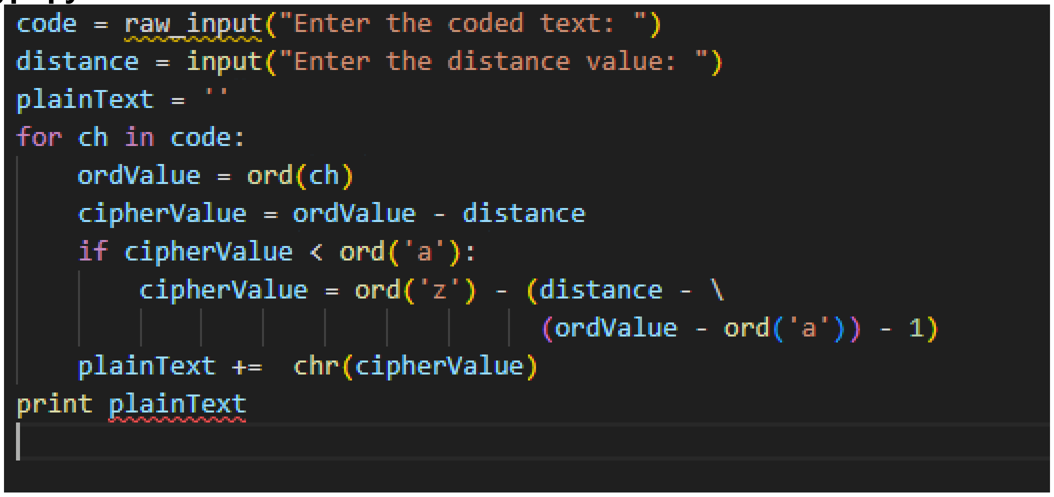
Figure 3.2 (Source code for decimaltobinary.py)



```
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$ python3 decimaltob
inary.py
Enter a decimal integer: 4
Quotient Remainder Binary
    2       0           0
    1       0          00
    0       1         100
The binary representation is 100
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$
```

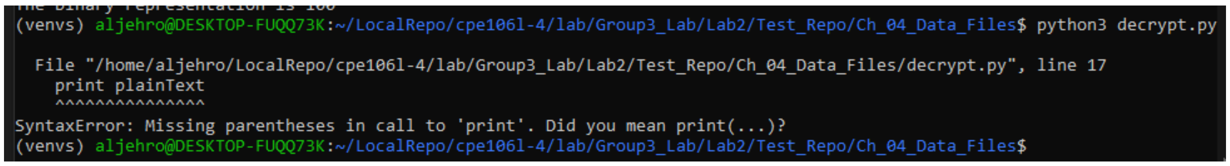
Figure 3.3 (The program decimaltobinary.py is working as intended)

#decrypt.py



```python
code = raw_input("Enter the coded text: ")
distance = input("Enter the distance value: ")
plainText = ''
for ch in code:
    ordValue = ord(ch)
    cipherValue = ordValue - distance
    if cipherValue < ord('a'):
        cipherValue = ord('z') - (distance - \
                                  (ordValue - ord('a')) - 1)
    plainText +=  chr(cipherValue)
print plainText
```

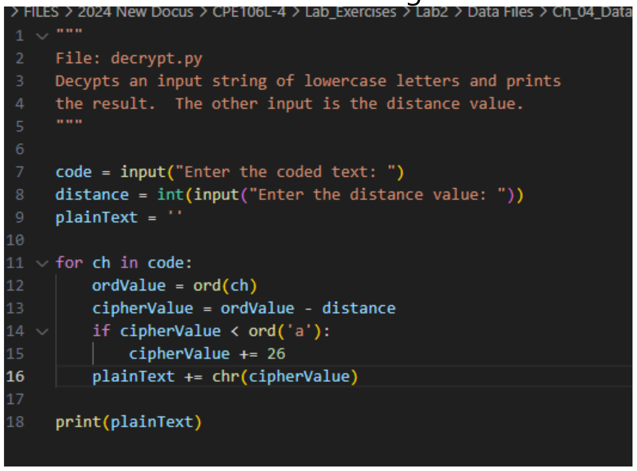
Figure 3.4 (Initial source code for decrypt.py. However its having syntax error)



```
The binary representation is 100
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$ python3 decrypt.py

  File "/home/aljehro/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files/decrypt.py", line 17
    print plainText
    ^^^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$
```

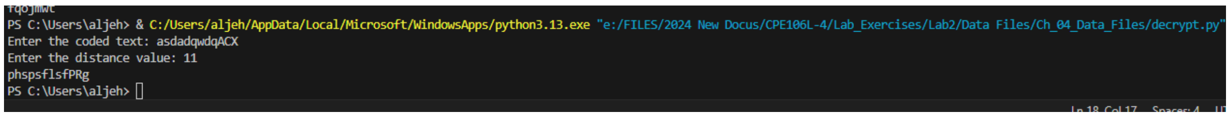
Figure 3.5 (Terminal window)

As seem in figure 3.5, there is really a syntax error and to fix this we rewrite print plainText to print(plainText). Below is the corrected code and debug terminal:


> FILES > 2024 New Docus > CPE106L-4 > Lab_Exercises > Lab2 > Data Files > Ch_04_Data

```python
"""
File: decrypt.py
Decypts an input string of lowercase letters and prints
the result.  The other input is the distance value.
"""

code = input("Enter the coded text: ")
distance = int(input("Enter the distance value: "))
plainText = ''

for ch in code:
    ordValue = ord(ch)
    cipherValue = ordValue - distance
    if cipherValue < ord('a'):
        cipherValue += 26
    plainText += chr(cipherValue)

print(plainText)
```

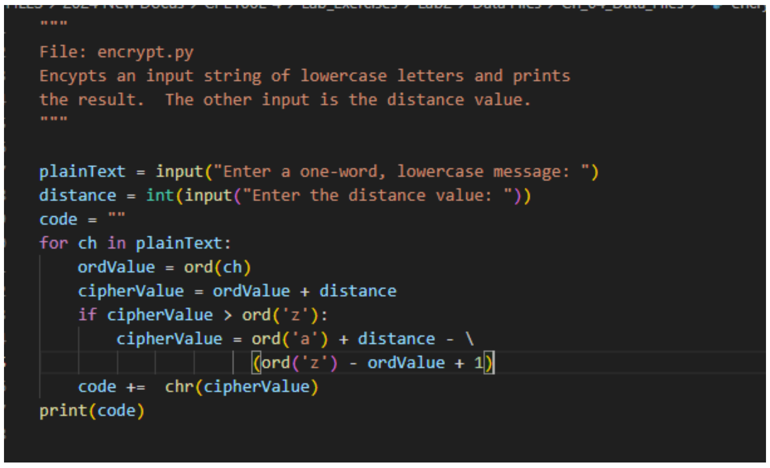
Figure 3.6.1 (Corrected decrypt.py)



```
fqojmwt
PS C:\Users\aljeh> & C:/Users/aljeh/AppData/Local/Microsoft/WindowsApps/python3.13.exe "e:/FILES/2024 New Docus/CPE106L-4/Lab_Exercises/Lab2/Data Files/Ch_04_Data_Files/decrypt.py"
Enter the coded text: asdadqwdqACX
Enter the distance value: 11
phspsflsfPRg
PS C:\Users\aljeh> 
```

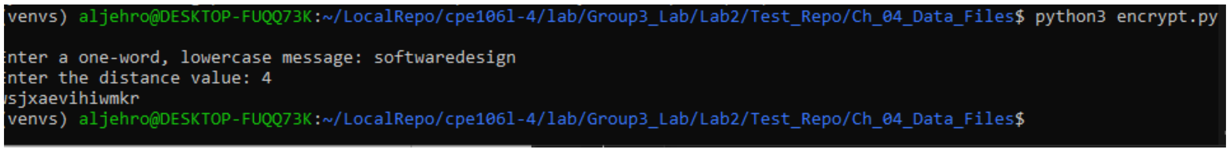
Figure 3.6.2 (working decrypt.py)

#encrypt.py



```python
"""
File: encrypt.py
Encypts an input string of lowercase letters and prints
the result.  The other input is the distance value.
"""

plainText = input("Enter a one-word, lowercase message: ")
distance = int(input("Enter the distance value: "))
code = ""
for ch in plainText:
    ordValue = ord(ch)
    cipherValue = ordValue + distance
    if cipherValue > ord('z'):
        cipherValue = ord('a') + distance - \
                      (ord('z') - ordValue + 1)
    code +=  chr(cipherValue)
print(code)
```

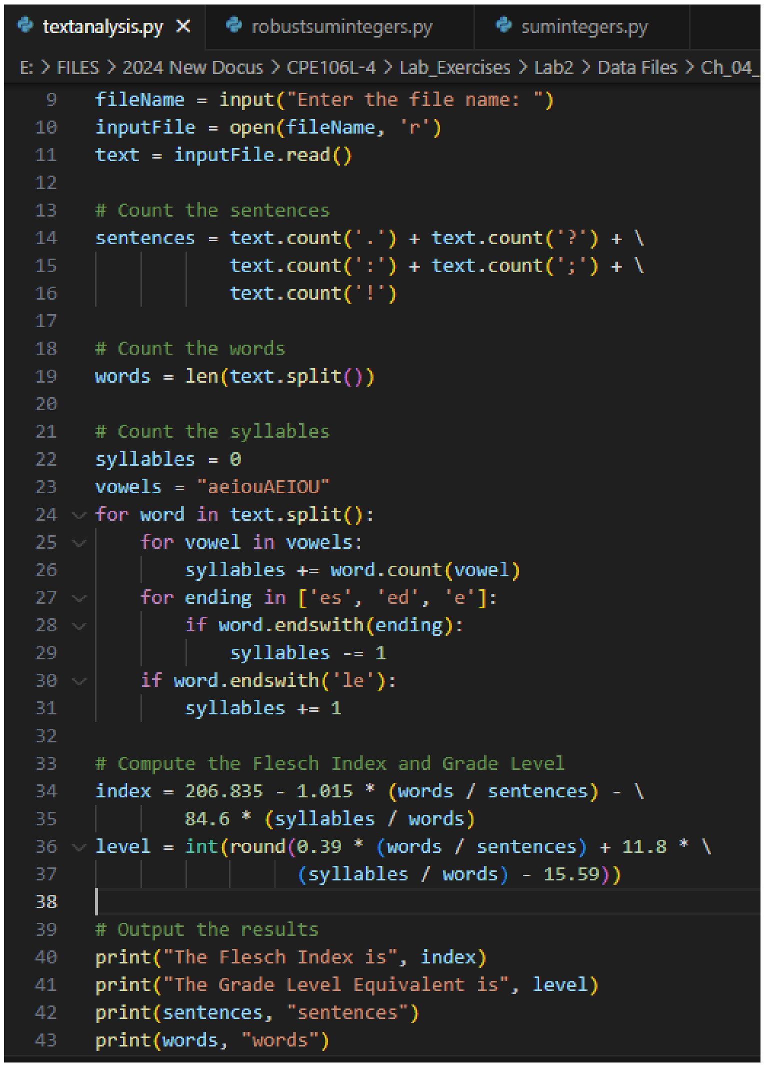
Figure 3.7.1 (encrypt.py)



```
venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$ python3 encrypt.py

nter a one-word, lowercase message: softwaredesign
nter the distance value: 4
sjxaevihiwmkr
venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$
```

Figure 3.7.2 (working encrypt.py)

#textanalysis.py

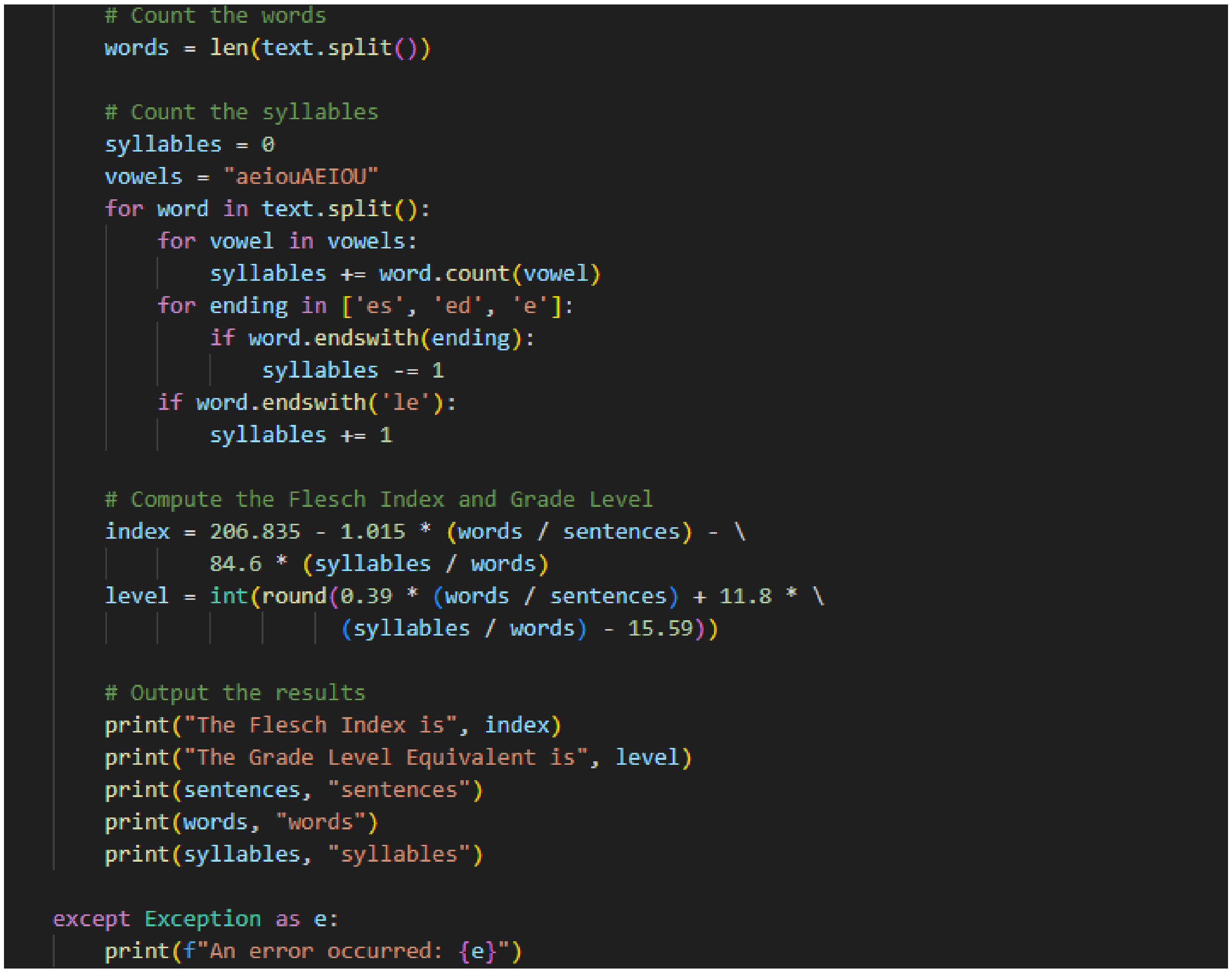


textanalysis.py
robustsumintegers.py
sumintegers.py
E: > FILES > 2024 New Docus > CPE106L-4 > Lab_Exercises > Lab2 > Data Files > Ch_04_

```python
fileName = input("Enter the file name: ")
inputFile = open(fileName, 'r')
text = inputFile.read()

# Count the sentences
sentences = text.count('.') + text.count('?') + \
            text.count(':') + text.count(';') + \
            text.count('!')

# Count the words
words = len(text.split())

# Count the syllables
syllables = 0
vowels = "aeiouAEIOU"
for word in text.split():
    for vowel in vowels:
        syllables += word.count(vowel)
    for ending in ['es', 'ed', 'e']:
        if word.endswith(ending):
            syllables -= 1
    if word.endswith('le'):
        syllables += 1

# Compute the Flesch Index and Grade Level
index = 206.835 - 1.015 * (words / sentences) - \
        84.6 * (syllables / words)
level = int(round(0.39 * (words / sentences) + 11.8 * \
                  (syllables / words) - 15.59))

# Output the results
print("The Flesch Index is", index)
print("The Grade Level Equivalent is", level)
print(sentences, "sentences")
print(words, "words")
```

Figure 3.8.1 (textanalysis.py)

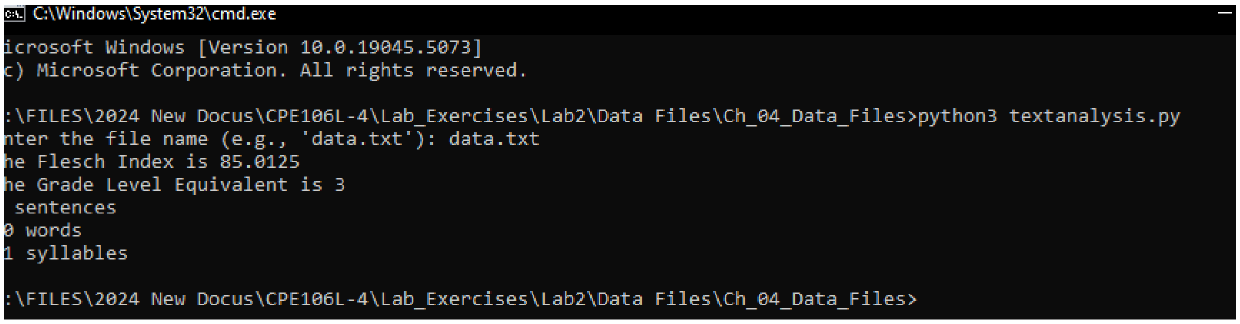
- Upon running the program, it would seem that the Python program is not reading a .text file even though both are located in the same folder. To fix this, here are some of the changes that were applied:



```
        # Count the words
        words = len(text.split())

        # Count the syllables
        syllables = 0
        vowels = "aeiouAEIOU"
        for word in text.split():
            for vowel in vowels:
                syllables += word.count(vowel)
            for ending in ['es', 'ed', 'e']:
                if word.endswith(ending):
                    syllables -= 1
            if word.endswith('le'):
                syllables += 1

        # Compute the Flesch Index and Grade Level
        index = 206.835 - 1.015 * (words / sentences) - \
                84.6 * (syllables / words)
        level = int(round(0.39 * (words / sentences) + 11.8 * \
                          (syllables / words) - 15.59))

        # Output the results
        print("The Flesch Index is", index)
        print("The Grade Level Equivalent is", level)
        print(sentences, "sentences")
        print(words, "words")
        print(syllables, "syllables")

    except Exception as e:
        print(f"An error occurred: {e}")
```

Figure 3.8.2 (fixed)

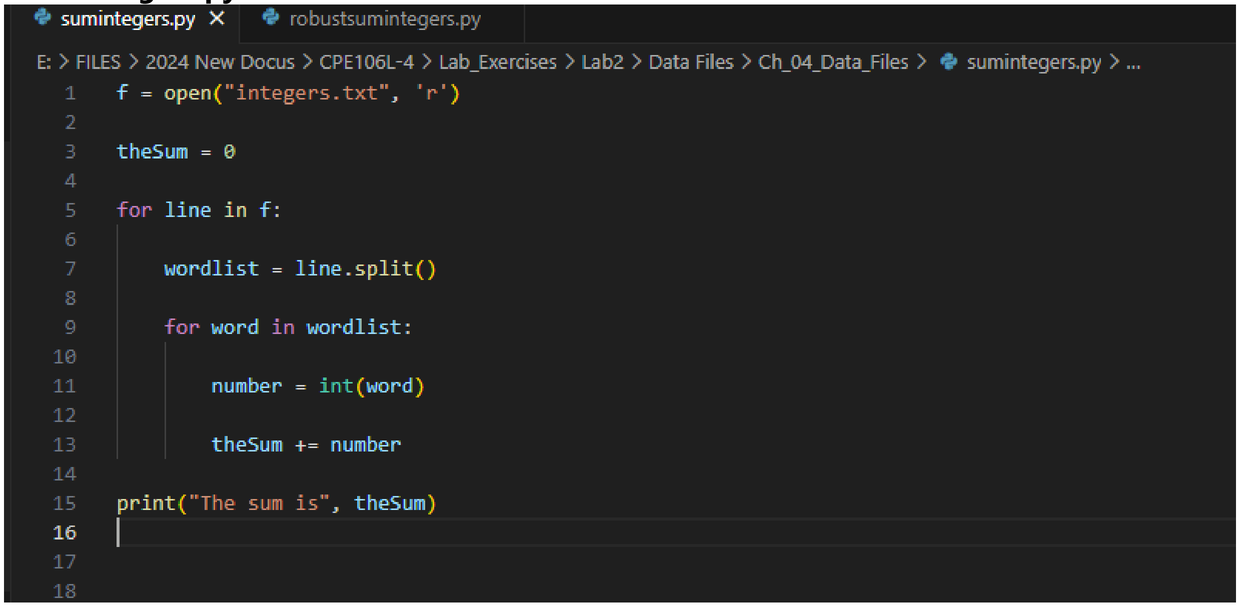
- For this fix, we added a coding line called import os in which the it allows a way of using operating system functionalities like reading or writing to the file system.



```
C:\Windows\System32\cmd.exe
icrosoft Windows [Version 10.0.19045.5073]
c) Microsoft Corporation. All rights reserved.

:\FILES\2024 New Docus\CPE106L-4\Lab_Exercises\Lab2\Data Files\Ch_04_Data_Files>python3 textanalysis.py
nter the file name (e.g., 'data.txt'): data.txt
he Flesch Index is 85.0125
he Grade Level Equivalent is 3
 sentences
0 words
1 syllables

:\FILES\2024 New Docus\CPE106L-4\Lab_Exercises\Lab2\Data Files\Ch_04_Data_Files>
```

Figure 3.8.3 (Output)

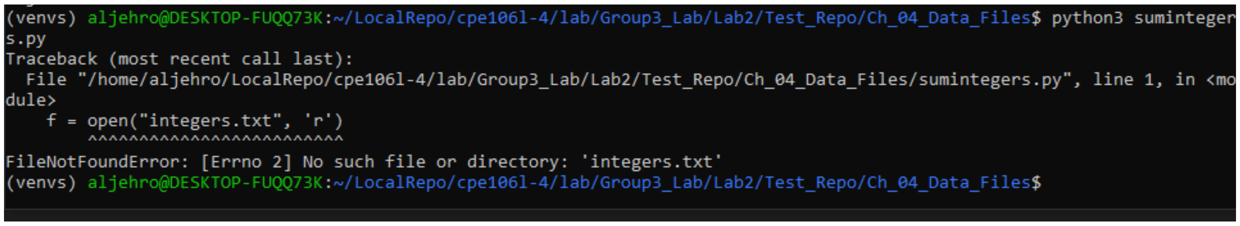
#sumintegers.py


sumintegers.py
robustsumintegers.py
E: > FILES > 2024 New Docus > CPE106L-4 > Lab_Exercises > Lab2 > Data Files > Ch_04_Data_Files > sumintegers.py > ...

```python
f = open("integers.txt", 'r')

theSum = 0

for line in f:

    wordlist = line.split()

    for word in wordlist:

        number = int(word)

        theSum += number

print("The sum is", theSum)
```

Figure 3.9.1 (sumintegers.py)

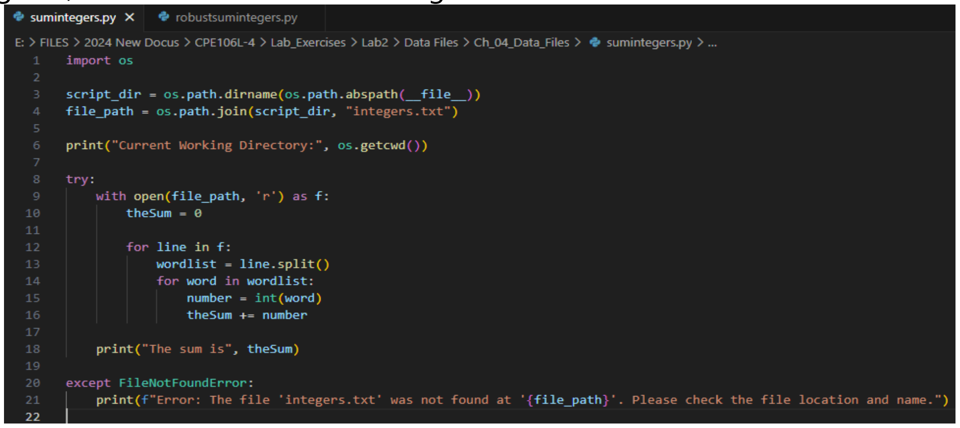
- We can observe here that the code is obviously lacking some important details such as the function to open a file. Below is the expected error:



```
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$ python3 suminteger
s.py
Traceback (most recent call last):
  File "/home/aljehro/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files/sumintegers.py", line 1, in <mo
dule>
    f = open("integers.txt", 'r')
        ^^^^^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'integers.txt'
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_04_Data_Files$
```

Figure 3.9.2 (sumintegers.py error)

- To debug this, we need to do the following:


sumintegers.py
robustsumintegers.py
E: > FILES > 2024 New Docus > CPE106L-4 > Lab_Exercises > Lab2 > Data Files > Ch_04_Data_Files > sumintegers.py > ...

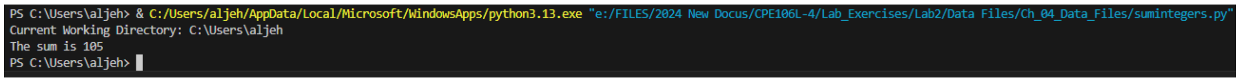
```python
import os

script_dir = os.path.dirname(os.path.abspath(__file__))
file_path = os.path.join(script_dir, "integers.txt")

print("Current Working Directory:", os.getcwd())

try:
    with open(file_path, 'r') as f:
        theSum = 0

        for line in f:
            wordlist = line.split()
            for word in wordlist:
                number = int(word)
                theSum += number

    print("The sum is", theSum)

except FileNotFoundError:
    print(f"Error: The file 'integers.txt' was not found at '{file_path}'. Please check the file location and name.")
```

Figure 3.9.3 (Fixed code for sumintegers.py)

- We apply the “os.path.abspath(__file__)” for it gets the absolute path of the current script file. What it does is that it ensures that the directory is known regardless of where the script is executed.

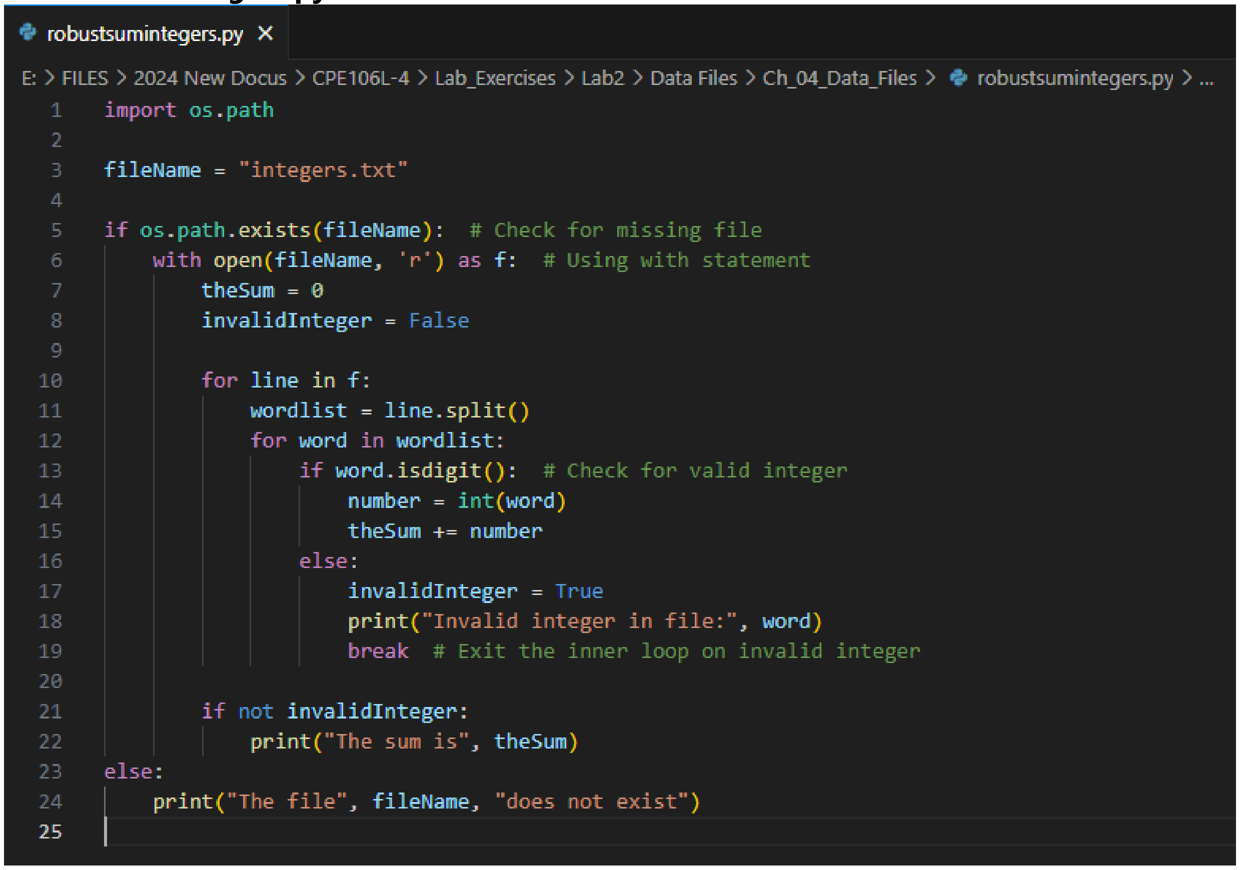
- “os.path.join(script_dir, "integers.txt")” creates a complete path to integers.txt, ensuring that it points to the correct file in the same directory as the script.

- The “with” statement is used to ensure that the file is properly closed after it is no longer needed. Below is the expected output of the corrected code:



```
PS C:\Users\aljeh> & C:/Users/aljeh/AppData/Local/Microsoft/WindowsApps/python3.13.exe "e:/FILES/2024 New Docus/CPE106L-4/Lab_Exercises/Lab2/Data Files/Ch_04_Data_Files/sumintegers.py"
Current Working Directory: C:\Users\aljeh
The sum is 105
PS C:\Users\aljeh>
```

#robustsumintegers.py



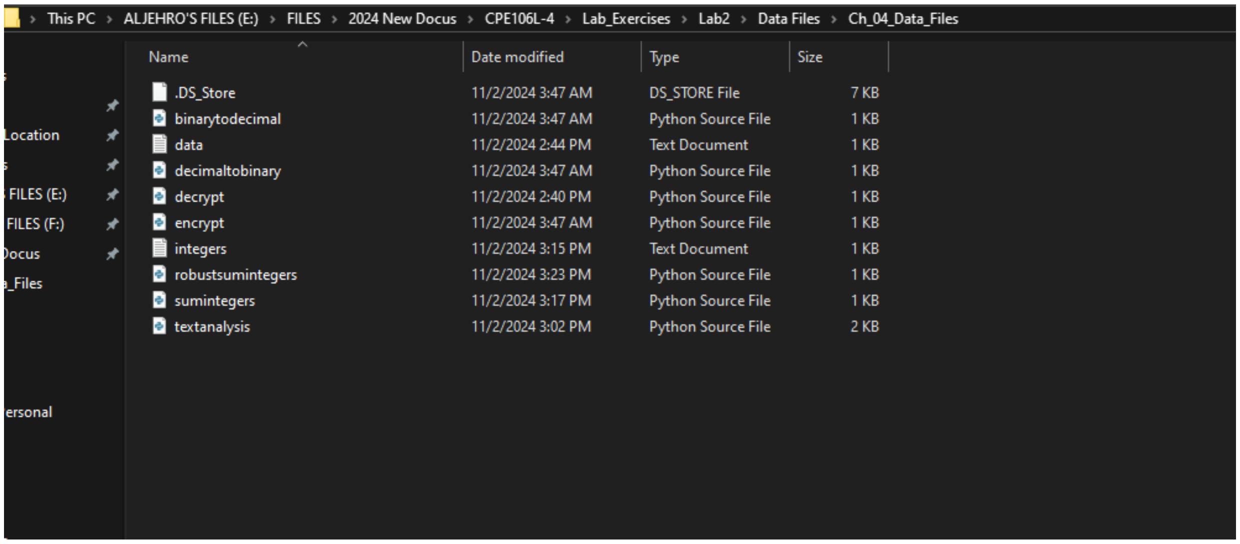
robustsumintegers.py

E: > FILES > 2024 New Docus > CPE106L-4 > Lab_Exercises > Lab2 > Data Files > Ch_04_Data_Files > robustsumintegers.py > ...

```python
import os.path

fileName = "integers.txt"

if os.path.exists(fileName):  # Check for missing file
    with open(fileName, 'r') as f:  # Using with statement
        theSum = 0
        invalidInteger = False

        for line in f:
            wordlist = line.split()
            for word in wordlist:
                if word.isdigit():  # Check for valid integer
                    number = int(word)
                    theSum += number
                else:
                    invalidInteger = True
                    print("Invalid integer in file:", word)
                    break  # Exit the inner loop on invalid integer

        if not invalidInteger:
            print("The sum is", theSum)
else:
    print("The file", fileName, "does not exist")

```

Figure 3.10.1

-This code checks if “integers.txt” exists in the current directory. To see if the code actually works, we go the folder where it is located:



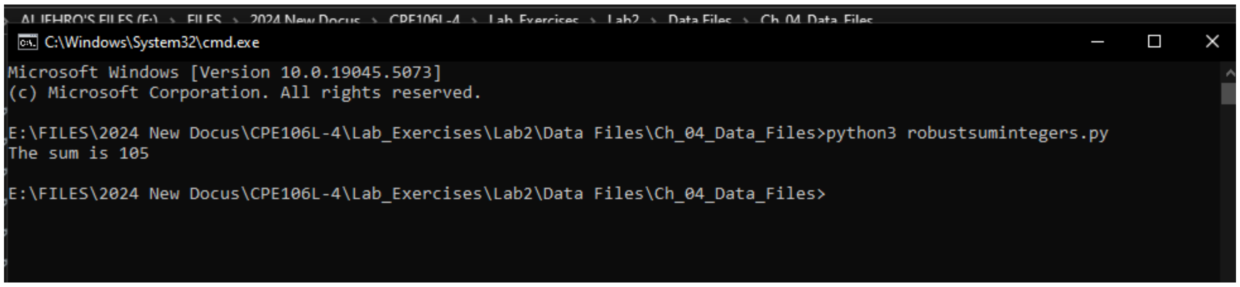
This PC › ALJEHRO'S FILES (E:) › FILES › 2024 New Docus › CPE106L-4 › Lab_Exercises › Lab2 › Data Files › Ch_04_Data_Files

Name	Date modified	Type	Size
.DS_Store	11/2/2024 3:47 AM	DS_STORE File	7 KB
binarytodecimal	11/2/2024 3:47 AM	Python Source File	1 KB
data	11/2/2024 2:44 PM	Text Document	1 KB
decimaltobinary	11/2/2024 3:47 AM	Python Source File	1 KB
decrypt	11/2/2024 2:40 PM	Python Source File	1 KB
encrypt	11/2/2024 3:47 AM	Python Source File	1 KB
integers	11/2/2024 3:15 PM	Text Document	1 KB
robustsumintegers	11/2/2024 3:23 PM	Python Source File	1 KB
sumintegers	11/2/2024 3:17 PM	Python Source File	1 KB
textanalysis	11/2/2024 3:02 PM	Python Source File	2 KB

Figure 3.10.2 (File Location of robustsumintegers.py)

This PC › ALJEHRO'S FILES (E:) › FILES › 2024 New Docus › CPE106L-4 › Lab_Exercises › Lab2 › Data Files › Ch_04_Data_Files

- Then, we enter "cmd" in this tab and proceed with the following



C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

E:\FILES\2024 New Docus\CPE106L-4\Lab_Exercises\Lab2\Data Files\Ch_04_Data_Files>python3 robustsumintegers.py
The sum is 105

E:\FILES\2024 New Docus\CPE106L-4\Lab_Exercises\Lab2\Data Files\Ch_04_Data_Files>
```

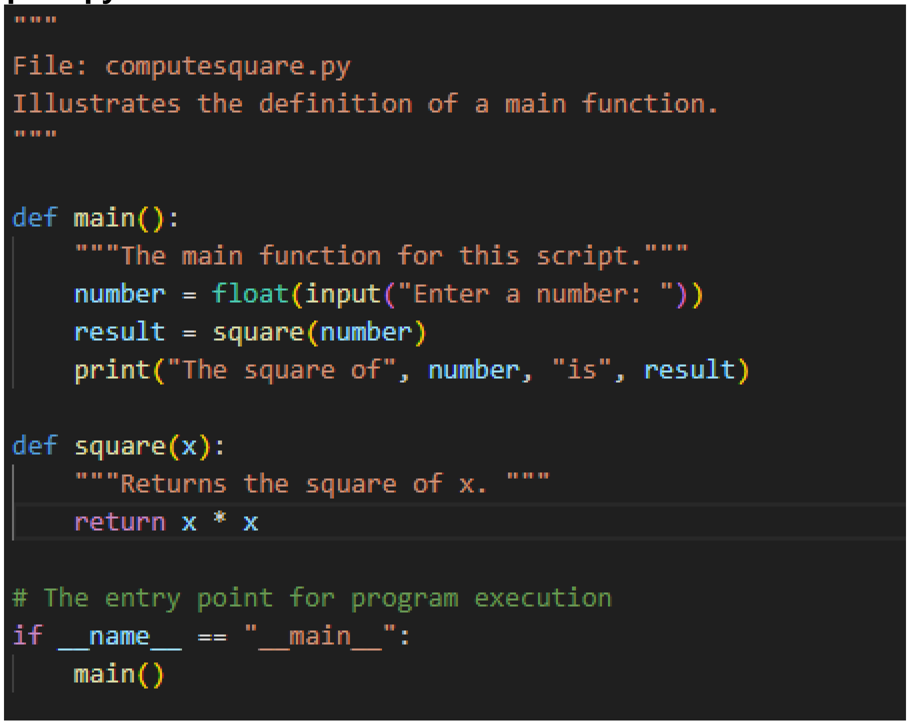
Figure 3.10.3 (run cmd robustsumintegers.py)

- We can confirm that the code is indeed working.

Chapter 5

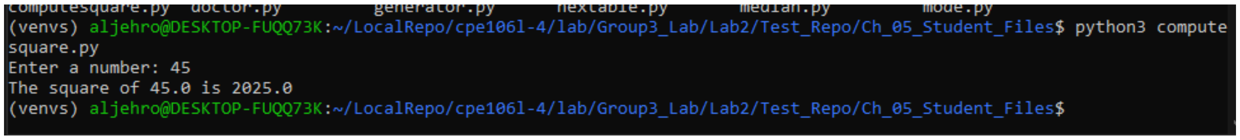
For the codes related to Chapter 5 these are the provided source codes and their outputs:

#computesquare.py



```python
"""
File: computesquare.py
Illustrates the definition of a main function.
"""

def main():
    """The main function for this script."""
    number = float(input("Enter a number: "))
    result = square(number)
    print("The square of", number, "is", result)

def square(x):
    """Returns the square of x. """
    return x * x

# The entry point for program execution
if __name__ == "__main__":
    main()
```

Figure 4.1.1 (computesquare.py)

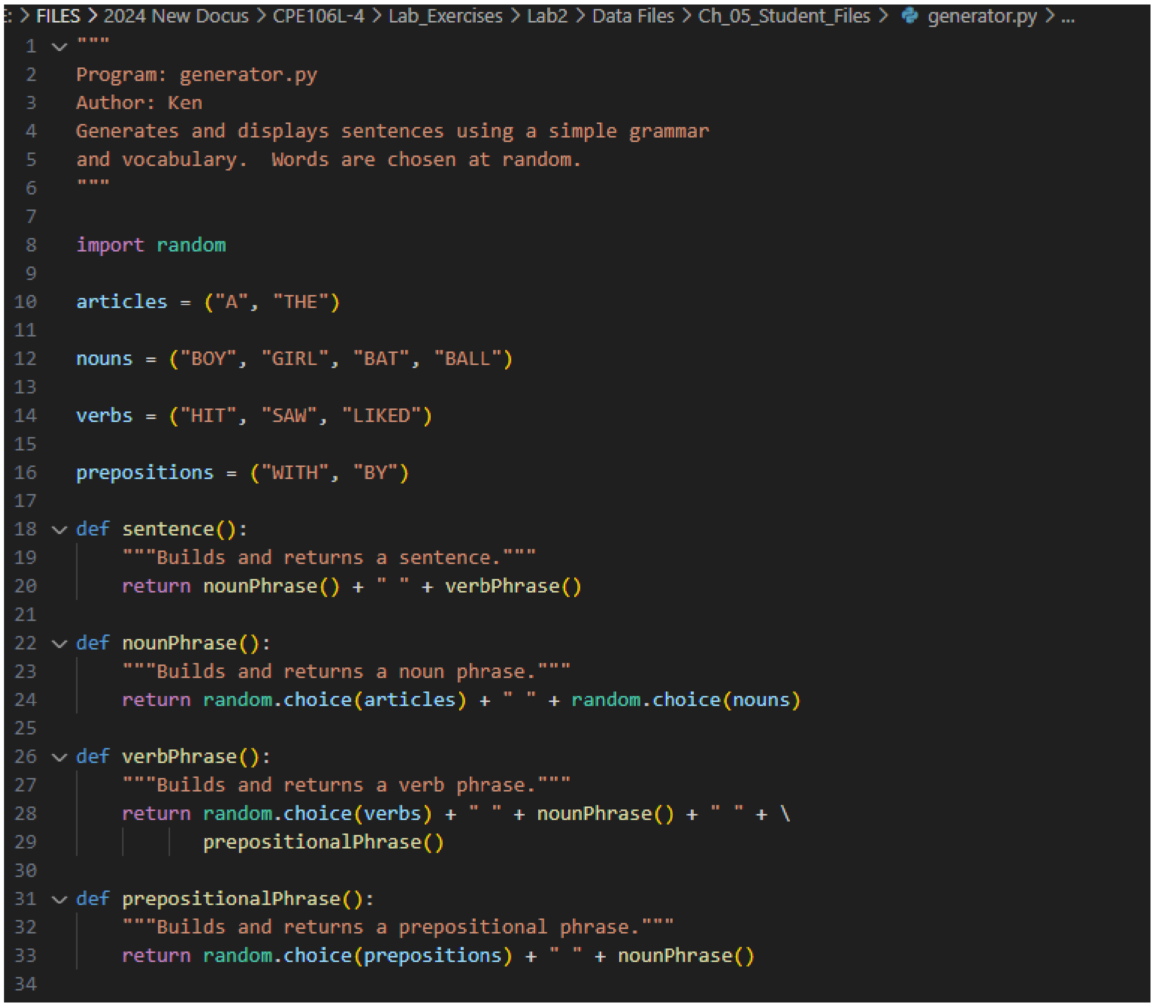


```
computesquare.py  doctor.py         generator.py      nextable.py       median.py         mode.py
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$ python3 compute
square.py
Enter a number: 45
The square of 45.0 is 2025.0
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$
```

Figure 4.1.2 (WSL terminal output)

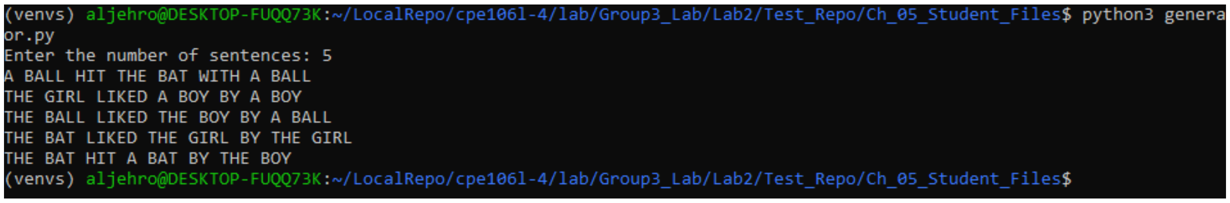
- The code defines a Python script that calculates the square of a user-provided number by first prompting for input, then passing that input to a square function that returns its square. And upon debugging, it is working properly.

#generator.py



```
: > FILES > 2024 New Docus > CPE106L-4 > Lab_Exercises > Lab2 > Data Files > Ch_05_Student_Files > generator.py > ...
"""
Program: generator.py
Author: Ken
Generates and displays sentences using a simple grammar
and vocabulary.  Words are chosen at random.
"""

import random

articles = ("A", "THE")

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

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

prepositions = ("WITH", "BY")

def sentence():
    """Builds and returns a sentence."""
    return nounPhrase() + " " + verbPhrase()

def nounPhrase():
    """Builds and returns a noun phrase."""
    return random.choice(articles) + " " + random.choice(nouns)

def verbPhrase():
    """Builds and returns a verb phrase."""
    return random.choice(verbs) + " " + nounPhrase() + " " + \
           prepositionalPhrase()

def prepositionalPhrase():
    """Builds and returns a prepositional phrase."""
    return random.choice(prepositions) + " " + nounPhrase()

```

Figure 4.2.1 (generator.py)



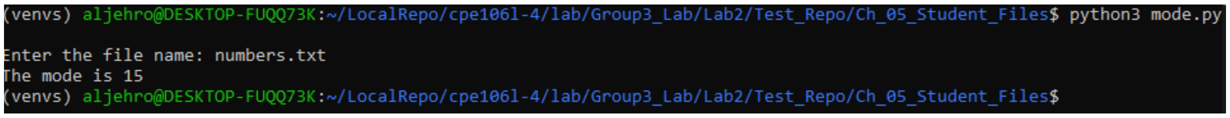
```
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$ python3 genera
or.py
Enter the number of sentences: 5
A BALL HIT THE BAT WITH A BALL
THE GIRL LIKED A BOY BY A BOY
THE BALL LIKED THE BOY BY A BALL
THE BAT LIKED THE GIRL BY THE GIRL
THE BAT HIT A BAT BY THE BOY
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$
```

Figure 4.2.2 (generator.py wsl terminal)

- This code defines a random sentence generator that constructs sentences using a simple grammar and vocabulary consisting of articles, nouns, verbs, and prepositions. It includes functions to build different parts of a sentence and prompts the user to specify how many sentences to generate, which are then printed to the terminal. Upon debugging, the code is confirmed to work.

```
#mode.py
fileName = input("Enter the file name: ")
f = open(fileName, 'r')

# Input the text, convert its to words to uppercase, and
# add the words to a list
words = []
for line in f:
    wordsInLine = line.split()
    for word in wordsInLine:
        words.append(word.upper())

# Obtain the set of unique words and their
# frequencies, saving these associations in
# a dictionary
theDictionary = {}
for word in words:
    number = theDictionary.get(word, None)
    if number == None:
        # word entered for the first time
        theDictionary[word] = 1
    else:
        # word already seen, increment its number
        theDictionary[word] = number + 1

# Find the mode by obtaining the maximum value
# in the dictionary and determining its key
theMaximum = max(theDictionary.values())
for key in theDictionary:
    if theDictionary[key] == theMaximum:
        print("The mode is", key)
        break
```

- The code reads a specified text file, converts all words to uppercase, and stores them in a list while counting their frequencies using a dictionary to associate each unique word with its occurrence count. It then determines the mode, or the most frequently occurring word, by finding the maximum value in the dictionary and prints the corresponding word. Below is the output in WSL terminal. This means that the code works as intended.

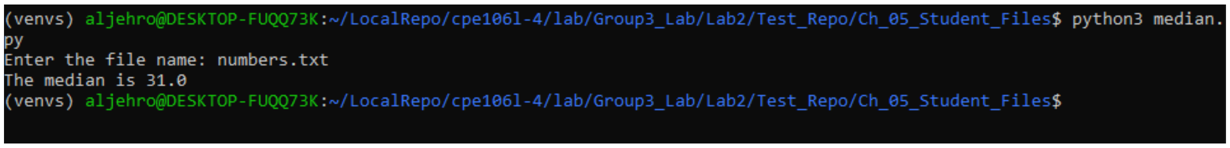


```
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$ python3 mode.py

Enter the file name: numbers.txt
The mode is 15
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$
```

Figure 4.3 (mode.py wsl terminal)

```
#median.py
fileName = input("Enter the file name: ")
f = open(fileName, 'r')

# Input the text, convert it to numbers, and
# add the numbers to a list
numbers = []
for line in f:
    words = line.split()
    for word in words:
        numbers.append(float(word))

# Sort the list and print the number at its midpoint
numbers.sort()
midpoint = len(numbers) // 2
print("The median is", end=" ")
if len(numbers) % 2 == 1:
    print(numbers[midpoint])
else:
    print((numbers[midpoint] + numbers[midpoint - 1]) / 2)
```

- Like how mode.py functions, the median.py reads a specified text file, converts all space-separated values to floating-point numbers, and stores them in a list**.** Then it proceeds to sort the list and calculates the median value and prints it. The output should be the middle number if the list length is odd. Below is the output:

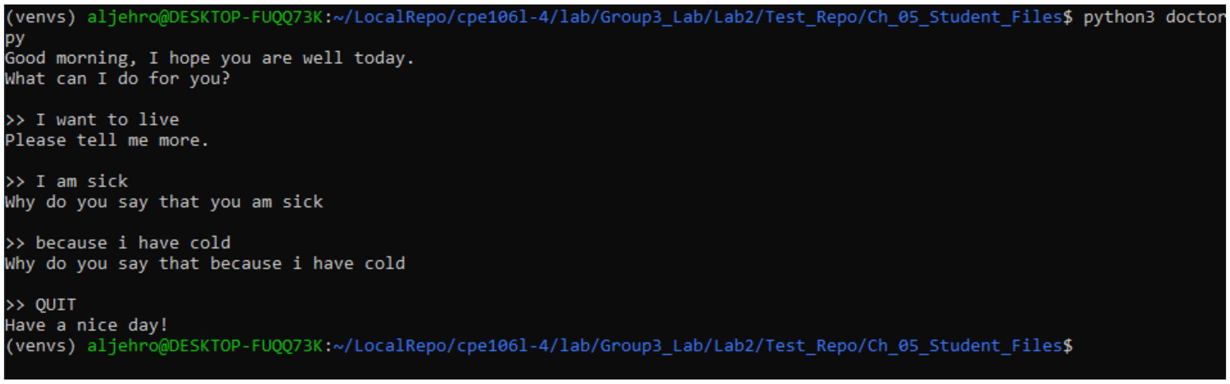


```
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$ python3 median.
py
Enter the file name: numbers.txt
The median is 31.0
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$
```

Figure 4.4 (median.py wsl terminal)

```
#doctor.py
import random

hedges = ("Please tell me more.",
        "Many of my patients tell me the same thing.",
        "Please coninue.")

qualifiers = ("Why do you say that ",
           "You seem to think that ",
           "Can you explain why ")

replacements = {"I":"you", "me":"you", "my":"your",
            "we":"you", "us":"you", "mine":"yours"}

def reply(sentence):
   """Implements two different reply strategies."""
   probability = random.randint(1, 4)
   if probability == 1:
      return random.choice(hedges)
   else:
      return random.choice(qualifiers) + changePerson(sentence)

def changePerson(sentence):
   """Replaces first person pronouns with second person
   pronouns."""
   words = sentence.split()
   replyWords = []
   for word in words:
      replyWords.append(replacements.get(word, word))
   return " ".join(replyWords)

def main():
   """Handles the interaction between patient and doctor."""
   print("Good morning, I hope you are well today.")
   print("What can I do for you?")
   while True:
      sentence = input("\n>> ")
      if sentence.upper() == "QUIT":
         print("Have a nice day!")
         break
      print(reply(sentence))

# The entry point for program execution
if __name__ == "__main__":
   main()
```

- This is a program simulates a simple interactive psychotherapy session where the user can converse with a virtual doctor. The doctor responds to user inputs with either hedging statements or qualifying questions while transforming first-person pronouns in the user's responses into second-person pronouns, encouraging the user to elaborate on their thoughts. Below is the output ran in the ubuntu terminal:

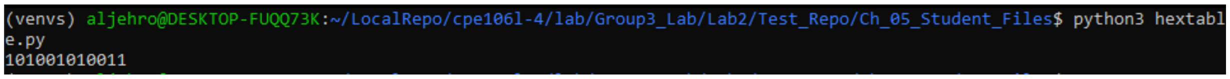


```
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$ python3 doctor
py
Good morning, I hope you are well today.
What can I do for you?

>> I want to live
Please tell me more.

>> I am sick
Why do you say that you am sick

>> because i have cold
Why do you say that because i have cold

>> QUIT
Have a nice day!
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe106l-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$
```

Figure 4.5 (doctor.py wsl terminal)

```
#hextable.py
hexToBinaryTable = {'0':'0000', '1':'0001', '2':'0010',
                    '3':'0011', '4':'0100', '5':'0101',
                    '6':'0110', '7':'0111', '8':'1000',
                    '9':'1001', 'A':'1010', 'B':'1011',
                    'C':'1100', 'D':'1101', 'E':'1110',
                    'F':'1111'}

def convert(number, table):
   """Builds and returns the base two representation of
   number. """
   binary = ""
   for digit in number:
      binary = table[digit] + binary
   return binary

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

- The code defines the hexToBinaryTable dictionary that maps hexadecimal digits to their corresponding 4-bit binary representations. The convert function takes a hexadecimal number (as a string) and the table as arguments, then constructs the binary representation by iterating over each digit in the input number, retrieving its binary equivalent from the table, and concatenating it to form the final binary string. Below is the expected output of the program:



```
(venvs) aljehro@DESKTOP-FUQQ73K:~/LocalRepo/cpe1061-4/lab/Group3_Lab/Lab2/Test_Repo/Ch_05_Student_Files$ python3 hextabl
e.py
101001010011
```

Figure 4.6 (hextable.py wsl terminal)

Overall comparison of Python to C++

-Python and C++ differ significantly in syntax, type systems, and memory management. While C++ necessitates explicit type declarations and manual memory management, which leads to more complex and verbose code, Python offers a simpler, more understandable syntax and dynamic typing, enabling rapid development and ease of use. Python uses exceptions for error handling, which makes it simpler to handle, while C++ uses try/catch blocks for a more structured method. C++ provides superior performance and control, making it the perfect choice for system development and resource-intensive applications, even if Python is favored for its large library and aptitude for scripting and data analysis.

PostLab

Programming Problems

1. Filename: stats.py

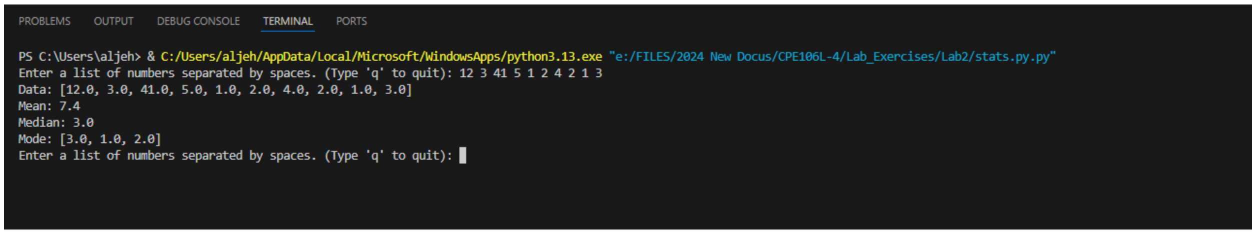
A group of statisticians at a local college has asked you to create a set of functions that compute the median and mode of a set of numbers, as defined in the below sample programs:

Define these functions in a module named stats.py. Also include a function named mean, which computes the average of a set of numbers. Each function should expect a list of numbers as an argument and return a single number. Each function should return 0 if the list is empty. Include a main function that tests the three statistical functions with a given list.

#stats.py

```
def mean(numbers):

  return sum(numbers) / len(numbers)

def median(numbers):
  sort_num = sorted(numbers)
  n = len(sort_num)
  mid = n // 2
  if n % 2 == 0:
    return (sort_num[mid - 1] + sort_num[mid]) / 2
  return sort_num[mid]

def mode(numbers):
  frequency = {}
  for number in numbers:
    frequency[number] = frequency.get(number, 0) + 1
  max_freq = max(frequency.values())
  modes = [k for k, v in frequency.items() if v == max_freq]
  return modes

def main():
  while True:
    user_input = input("Enter a list of numbers separated by spaces. (Type 'q' to
quit): ")
    if user_input.lower() == 'q':
      print("Terminating program.....")
      break
    try:
      data = list(map(float, user_input.split()))
      print("Data:", data)
      print("Mean:", mean(data))
      print("Median:", median(data))
      print("Mode:", mode(data))
    except ValueError:
      print("Invalid input.")

if __name__ == "__main__":
  main()
```

Output:



```
PS C:\Users\aljeh> & C:/Users/aljeh/AppData/Local/Microsoft/WindowsApps/python3.13.exe "e:/FILES/2024 New Docus/CPE106L-4/Lab_Exercises/Lab2/stats.py.py"
Enter a list of numbers separated by spaces. (Type 'q' to quit): 12 3 41 5 1 2 4 2 1 3
Data: [12.0, 3.0, 41.0, 5.0, 1.0, 2.0, 4.0, 2.0, 1.0, 3.0]
Mean: 7.4
Median: 3.0
Mode: [3.0, 1.0, 2.0]
Enter a list of numbers separated by spaces. (Type 'q' to quit):
```

2. Filename: LR2_2.py

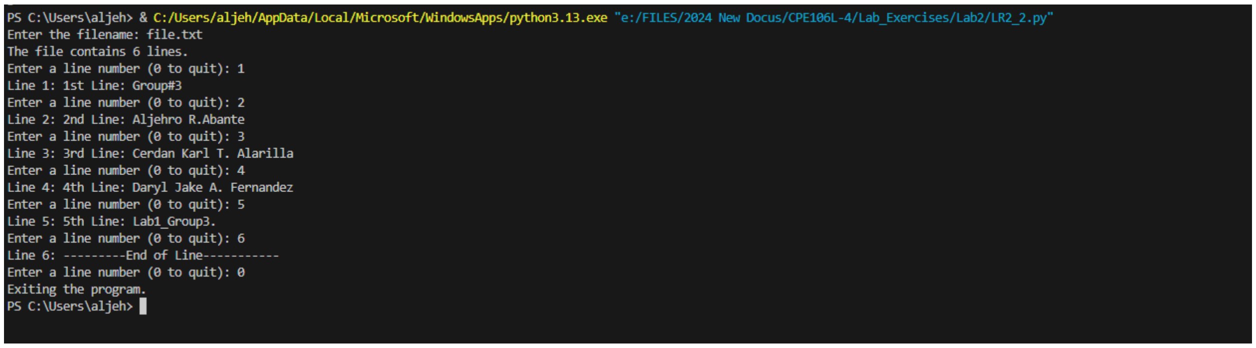
Write a program that allows the user to navigate the lines of text in a file. The program should prompt the user for a filename and input the lines of text into a list. The program then enters a loop in which it prints the number of lines in the file and prompts the user for a line number. Actual line numbers range from 1 to the number of lines in the file. If the input is 0, the program quits. Otherwise, the program prints the line associated with that number.

#LR2_2.py

import os

```
def navigate_file():
  filename = input("Enter the filename: ")

  current_dir = os.path.dirname(os.path.abspath(__file__))
  file_path = os.path.join(current_dir, filename)

  try:
    with open(file_path, 'r') as file:
      lines = file.readlines()

    num_lines = len(lines)
    print(f"The file contains {num_lines} lines.")
    while True:
      try:
        line_num = int(input("Enter a line number (0 to quit): "))
      except ValueError:
        print("Please enter a valid number.")
        continue

      if line_num == 0:
        print("Exiting the program.")
        break
      elif 1 <= line_num <= num_lines:
        print(f"Line {line_num}: {lines[line_num - 1].strip()}")
      else:
        print(f"Please enter a line number between 1 and {num_lines}.")

  except FileNotFoundError:
    print(f"The file '{filename}' was not found in the current directory.")
  except IOError:
    print("An error occurred while reading the file.")

navigate_file()
```

Output:



```
PS C:\Users\aljeh> & C:/Users/aljeh/AppData/Local/Microsoft/WindowsApps/python3.13.exe "e:/FILES/2024 New Docus/CPE106L-4/Lab_Exercises/Lab2/LR2_2.py"
Enter the filename: file.txt
The file contains 6 lines.
Enter a line number (0 to quit): 1
Line 1: 1st Line: Group#3
Enter a line number (0 to quit): 2
Line 2: 2nd Line: Aljehro R.Abante
Enter a line number (0 to quit): 3
Line 3: 3rd Line: Cerdan Karl T. Alarilla
Enter a line number (0 to quit): 4
Line 4: 4th Line: Daryl Jake A. Fernandez
Enter a line number (0 to quit): 5
Line 5: 5th Line: Lab1_Group3.
Enter a line number (0 to quit): 6
Line 6: ---------End of Line----------
Enter a line number (0 to quit): 0
Exiting the program.
PS C:\Users\aljeh> 
```

3. name: generator_modified.py

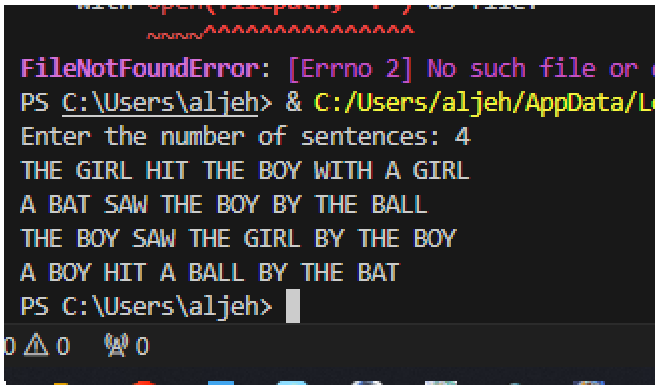
Modify the sentence-generator program of Case Study 5.3 so that it inputs its vocabulary from a set of text files at startup. The filenames are nouns.txt, verbs. txt, articles.txt, and prepositions.txt. (Hint: Define a single new function, getWords. This function should expect a filename as an argument. The function should open an input file with this name, define a temporary list, read words from the file, and add them to the list. The function should then convert the list to a tuple and return this tuple. Call the function with an actual filename to initialize each of the four variables for the vocabulary.)

generator_modified.py

```
import random
import os

def getWords(filename):
  """Reads words from a file and returns them as a tuple."""
  current_dir = os.path.dirname(os.path.abspath(__file__))
  filepath = os.path.join(current_dir, filename)

  with open(filepath, 'r') as file:
    words = []
    for line in file:
      words.extend(line.strip().split())
    return tuple(words)

articles = getWords("articles.txt")
nouns = getWords("nouns.txt")
verbs = getWords("verbs.txt")
prepositions = getWords("prepositions.txt")

def sentence():
  """Builds and returns a sentence."""
  return nounPhrase() + " " + verbPhrase()

def nounPhrase():
  """Builds and returns a noun phrase."""
  return random.choice(articles) + " " + random.choice(nouns)

def verbPhrase():
  """Builds and returns a verb phrase."""
  return random.choice(verbs) + " " + nounPhrase() + " " + prepositionalPhrase()

def prepositionalPhrase():
  """Builds and returns a prepositional phrase."""
  return random.choice(prepositions) + " " + nounPhrase()

def main():
  """Allows the user to input the number of sentences
  to generate."""
  number = int(input("Enter the number of sentences: "))
  for count in range(number):
    print(sentence())

if __name__ == "__main__":
```

```
main()
```

Output:



```
            ~~~~^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or
PS C:\Users\aljeh> & C:/Users/aljeh/AppData/L
Enter the number of sentences: 4
THE GIRL HIT THE BOY WITH A GIRL
A BAT SAW THE BOY BY THE BALL
THE BOY SAW THE GIRL BY THE BOY
A BOY HIT A BALL BY THE BAT
PS C:\Users\aljeh>
```