Table of Contents

[Group Profile 2](#_Toc156169783)

[Question1: Use the open-source NLP 3](#_Toc156169784)

[Task 1: Extract the ‘text’ in all the CSV files and store into a single ‘.txt file’ 3](#_Toc156169785)

[Task 2: Research 4](#_Toc156169786)

[Task 3: Programming and Research 6](#_Toc156169787)

[Task 4: Named-Entity Recognition (NER) 11](#_Toc156169788)

[Question 2 18](#_Toc156169789)

[Chapter 1: The Gatekeeper 18](#_Toc156169790)

[Chapter 2: The Chamber of Strings 21](#_Toc156169791)

[First program: 21](#_Toc156169792)

[Second program: 22](#_Toc156169793)

[Question 3 23](#_Toc156169794)

[1. Fixing the next code will reveal the key. 23](#_Toc156169795)

[2. Write the decryption function to decrypt the ‘encrypted code’ to the original code. 24](#_Toc156169796)

[3. Correct the errors and provide the comments 26](#_Toc156169797)

[Question 4 28](#_Toc156169798)

# 

# Group Profile

**Group Members:**  
[Sameer Basnet] - [S372941]  
[Samir Dhakal] - [S373048]  
[Susanti Djie] - [S375655]  
[Nishat Anjum] - [S374044]

**Github:**

<https://github.com/Sameer84/HIT137-Assignment-02.git>

# Question1: Use the open-source NLP

## Task 1: Extract the ‘text’ in all the CSV files and store into a single ‘.txt file’

**Programming code:**

# 1-1 Extract file

# Extract the ‘text’ in all the CSV files

# and store them into a single ‘.txt file’.

try:

# put csv file names to list

csv\_files = ['./q1/CSV1.csv', './q1/CSV2.csv', './q1/CSV3.csv', './q1/CSV4.csv']

with open('./q1/merged\_csv.txt', 'w') as f\_out:

# open each csv

for file in csv\_files:

with open(file, 'r') as f\_in:

content = f\_in.read()

f\_out.write(content)

print()

print('proccess done.')

except IOError as e:

# if any csv file is missing

print('IOError : ', e.errno, e)

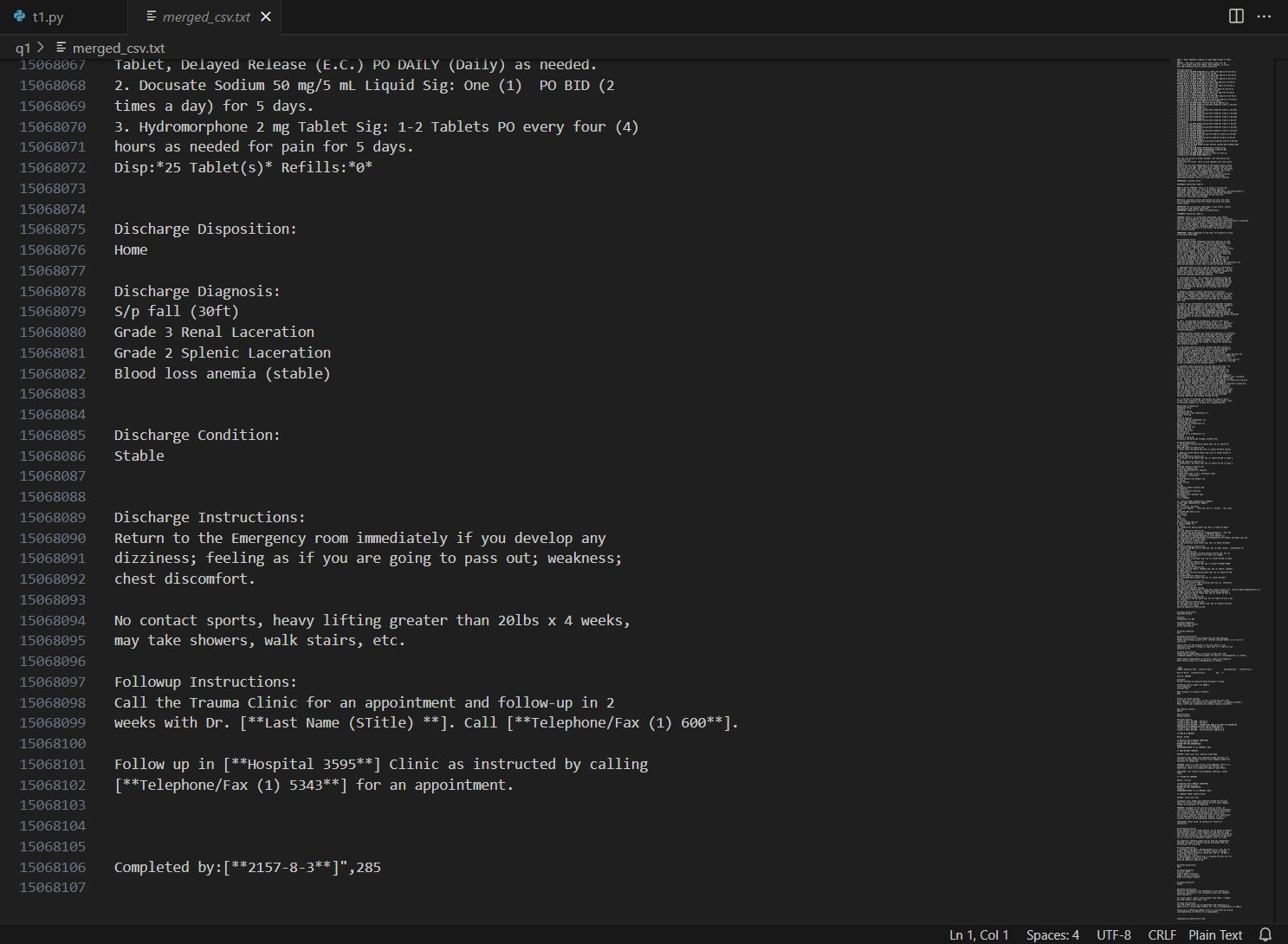
finally:

print('clean up...')

f\_in.close()

f\_out.close()

**#Output**



## Task 2: Research

**Programming code:**

# requirements.txt

pandas

numpy

torch

# spacy and en\_core\_web\_sm model

spacy==3.6.1

# !!important: scispacy only works with python 3.8 or below with nmslib

# check: <https://github.com/allenai/scispacy/issues/291>

# scispacy and en\_ner\_bc5cdr\_md model

scispacy==0.5.3

<https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.5.3/en_core_sci_sm-0.5.3.tar.gz>

<https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.5.3/en_ner_bc5cdr_md-0.5.3.tar.gz>

# transformers

transformers

**#Output**

pip install -r .\requirements.txt

## Task 3: Programming and Research

**Programming code:**

import os

import shutil

import csv

from itertools import islice

from collections import Counter

from transformers import AutoTokenizer

NUM\_OF\_LINES = 100000

WORKER\_POOL\_SIZE = 2

PRE\_DIR\_NAME = './q1'

INPUT\_FILE = PRE\_DIR\_NAME + '/merged\_csv.txt'

CHUNKS\_DIRECTORY = PRE\_DIR\_NAME + '/chunks'

def delete\_dir\_if\_exists(dir\_name):

if os.path.exists(dir\_name):

shutil.rmtree(dir\_name)

def create\_dir\_if\_not\_exists(dir\_name):

if not os.path.exists(dir\_name):

os.makedirs(dir\_name)

def slice\_huge\_file():

cnt = 0

with open(INPUT\_FILE) as f:

while True:

next\_n\_lines = list(islice(f, NUM\_OF\_LINES))

cnt += 1

if not next\_n\_lines:

break

with open(CHUNKS\_DIRECTORY + '/sub\_huge\_{}.txt'.format(cnt), 'w') as out:

out.writelines(next\_n\_lines)

def count\_file\_words(input\_file):

count\_from\_at = count\_and\_top\_words(input\_file, "dmis-lab/biobert-v1.1")

with open(input\_file, 'r') as f:

lines = [line.strip() for line in f.readlines() if line.strip()]

count\_from\_bl = Counter(lines)

return count\_from\_bl, count\_from\_at

def count\_unique\_tokens(file\_path):

tokenizer = AutoTokenizer.from\_pretrained("dmis-lab/biobert-v1.1")

# Read the text file

with open(file\_path, 'r') as file:

text = file.read()

# Tokenize the text

tokens = tokenizer.tokenize(text)

# Count the unique tokens

token\_counts = Counter(tokens)

# Get the top 30 words

top\_30\_words = token\_counts.most\_common(30)

return top\_30\_words

def count\_and\_top\_words(input\_file, model\_name, chunk\_size=750):

# Load the AutoTokenizer

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

# Initialize an empty counter for token counts

token\_counts = Counter()

# Read the text from the file in chunks

with open(input\_file, 'r', encoding='utf-8') as file:

while True:

chunk = file.read(chunk\_size)

if not chunk:

break

# Tokenize the chunk and update token counts

tokens = tokenizer.tokenize(tokenizer.decode(tokenizer.encode(chunk)))

token\_counts.update(tokens)

return token\_counts

def export\_to\_file(file\_path, most\_common\_30):

if os.path.exists(file\_path):

os.remove(file\_path)

with open(file\_path, 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['Word', 'Count'])

writer.writerows(most\_common\_30)

file.close()

if \_\_name\_\_ == '\_\_main\_\_':

delete\_dir\_if\_exists(CHUNKS\_DIRECTORY)

create\_dir\_if\_not\_exists(CHUNKS\_DIRECTORY)

slice\_huge\_file()

sub\_files = [os.path.join(CHUNKS\_DIRECTORY, f) for f in os.listdir(CHUNKS\_DIRECTORY) if f.startswith('sub\_huge')]

# results from built in library

results\_bl = Counter()

# results from Auto Tokenizer

results\_at = Counter()

for sub\_file in sub\_files:

count\_from\_bl, count\_from\_at = count\_file\_words(sub\_file)

results\_bl += count\_from\_bl

results\_at += count\_from\_at

most\_common30\_bl = results\_bl.most\_common(30)

most\_common30\_at = results\_at.most\_common(30)

# Export most common 30 words from built in library to CSV

export\_to\_file(PRE\_DIR\_NAME + '/most\_common\_30\_builtInLibrary.csv', most\_common30\_bl)

# Export most common 30 words from Auto Tokenizer to txt

export\_to\_file(PRE\_DIR\_NAME + '/most\_common\_30\_AutoTokenizer.txt', most\_common30\_at)

# clean up

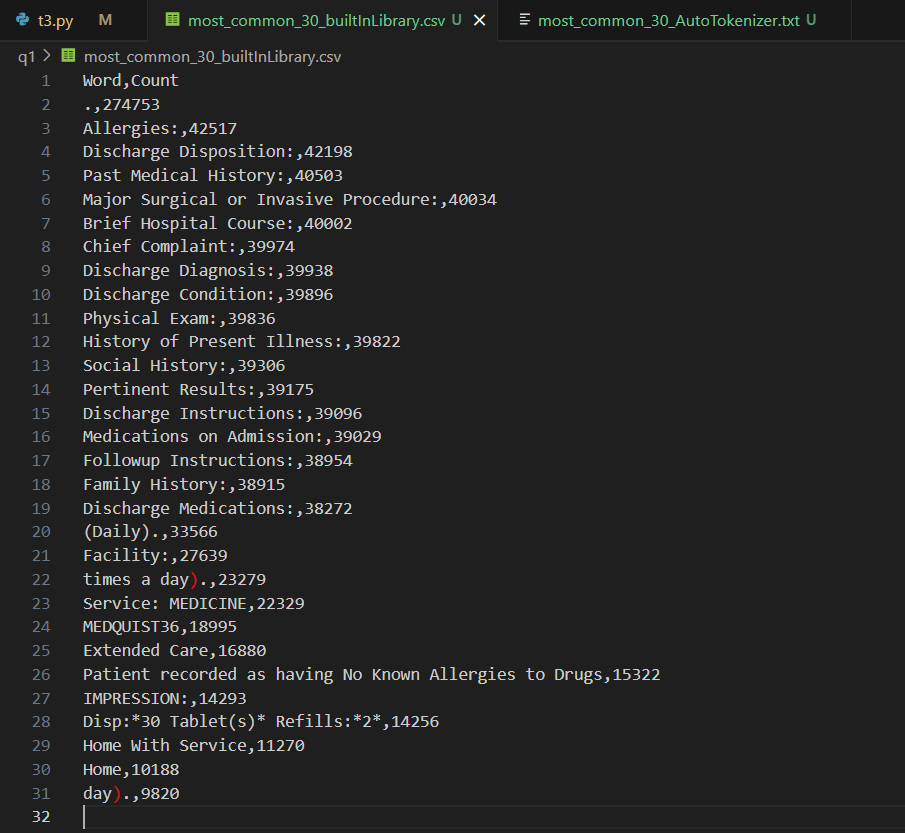
results\_bl.clear()

results\_at.clear()

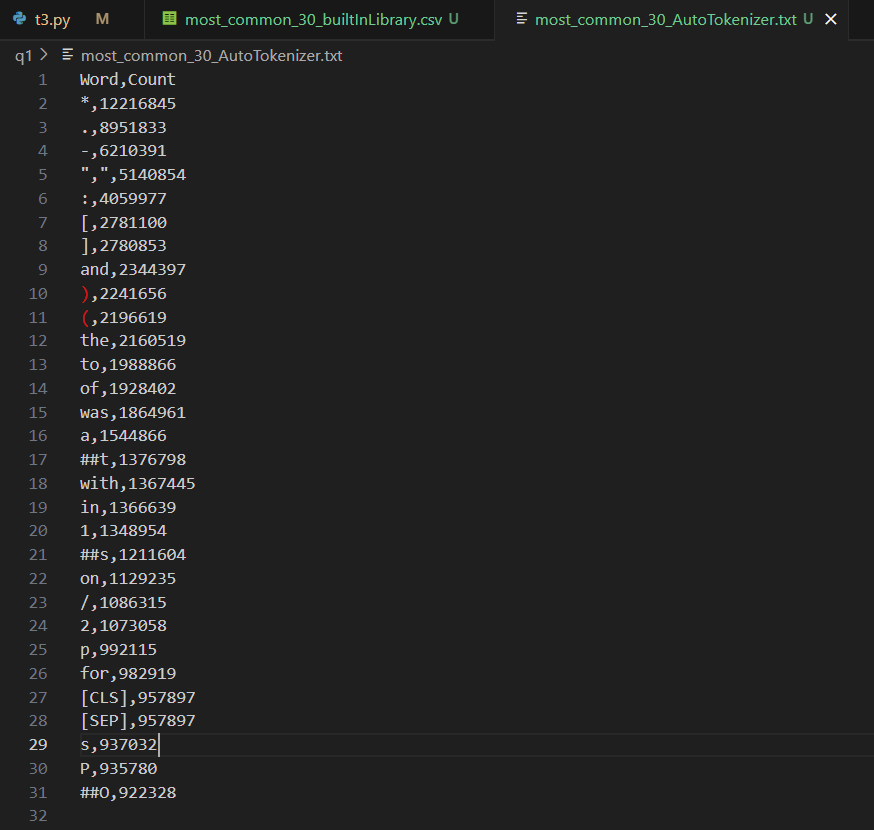
**#Output**

Using any in-built library present in Python, count the occurrences of the words in the text (.txt) and give the ‘Top 30’ most common words. And store the ‘Top 30’ common words and their counts into a CSV file.

most\_common\_30\_builtInLibrary.csv



Using the ‘Auto Tokenizer’ function in the ‘Transformers’ library, write a ‘function’ to count unique tokens in the text (.txt) and give the ‘Top 30’ words.

most\_common\_30\_AutoTokenizer.txt

## Task 4: Named-Entity Recognition (NER)

Extract the ‘diseases’, and ‘drugs’ entities in the ‘.txt file’ separately using ‘en\_core\_sci\_sm’/’en\_ner\_bc5cdr\_md’ and biobert. And compare the differences between the two models (Example: Total entities detected by both of them, what’s the difference, check for most common words, and check the difference.)

**Programming code:**

import os

from collections import Counter

import spacy, scispacy

from transformers import AutoTokenizer, AutoModelForTokenClassification, pipeline

PRE\_DIR\_NAME = './q1'

INPUT\_FILE = PRE\_DIR\_NAME + '/merged\_csv.txt'

OUTPUT\_DIR = PRE\_DIR\_NAME + '/results'

MODEL\_NAME = 'dmis-lab/biobert-v1.1'

def count\_and\_top\_words(doc, chunk\_size=900000, top\_n=30):

entities = [ent.text for ent in doc.ents]

total\_entities = len(entities)

# Extract 'diseases' entities

diseases\_ent = [ent.text for ent in doc.ents if ent.label\_ == 'DISEASE']

# Extract 'drugs' entities

drugs\_ent = [ent.text for ent in doc.ents if ent.label\_ == 'DRUG']

# Extract entities length

extract\_entities = len(diseases\_ent) + len(drugs\_ent)

# Tokenize the chunk and update token counts

top\_tokens = Counter(entities)

return entities, total\_entities, diseases\_ent, drugs\_ent, extract\_entities, top\_tokens

def run():

if not os.path.exists(OUTPUT\_DIR):

os.makedirs(OUTPUT\_DIR)

# Load the 'en\_core\_sci\_sm' model

en\_core\_sci\_sm = "en\_core\_sci\_sm"

nlp\_sci\_sm = spacy.load(en\_core\_sci\_sm)

entities\_sci\_sm = []

total\_entities\_sci\_sm = 0

diseases\_model\_sci\_sm = []

drugs\_model\_sci\_sm = []

tokens\_sci\_sm = Counter()

# Load the 'en\_ner\_bc5cdr\_md' model

en\_ner\_bc5cdr\_md = "en\_ner\_bc5cdr\_md"

nlp\_bc5cdr\_md = spacy.load(en\_ner\_bc5cdr\_md)

entities\_bc5cdr\_md = []

total\_entities\_bc5cdr\_md = 0

diseases\_model\_bc5cdr\_md = []

drugs\_model\_bc5cdr\_md = []

tokens\_bc5cdr\_md = Counter()

# Load the NER pipeline with biobert model

tokenizer = AutoTokenizer.from\_pretrained(MODEL\_NAME)

model = AutoModelForTokenClassification.from\_pretrained(MODEL\_NAME)

# Extract entities using the NER pipeline

nlp = pipeline("ner", model=model, tokenizer=tokenizer)

# Load the 'en\_core\_sci\_sm' model

entities\_biobert = []

total\_biobert = 0

filtered\_biobert = []

tokens\_biobert = Counter()

with open(OUTPUT\_DIR + '/t4\_result.txt', 'w') as f\_out:

with open(INPUT\_FILE, 'r', encoding='utf-8') as file:

# Read the text from the file in chunks

chunk\_size = 900000

while True:

chunk = file.read(chunk\_size)

if not chunk:

break

entities, total\_entities, diseases\_ent, drugs\_ent, extract\_entities, top\_tokens = \

count\_and\_top\_words(nlp\_sci\_sm(chunk), chunk\_size=chunk\_size)

entities\_sci\_sm += entities

total\_entities\_sci\_sm += total\_entities

diseases\_model\_sci\_sm += diseases\_ent

drugs\_model\_sci\_sm += drugs\_ent

tokens\_sci\_sm += top\_tokens

entities, total\_entities, diseases\_ent, drugs\_ent, extract\_entities, top\_tokens = \

count\_and\_top\_words(nlp\_bc5cdr\_md(chunk), chunk\_size=chunk\_size)

entities\_bc5cdr\_md += entities

total\_entities\_bc5cdr\_md += total\_entities

diseases\_model\_bc5cdr\_md += diseases\_ent

drugs\_model\_bc5cdr\_md += drugs\_ent

tokens\_bc5cdr\_md += top\_tokens

print(f'Total entities detected {en\_core\_sci\_sm}: {total\_entities\_sci\_sm}')

print(f'Total entities detected {en\_ner\_bc5cdr\_md}: {total\_entities\_bc5cdr\_md}')

f\_out.write(f'Total entities detected {en\_core\_sci\_sm}: {total\_entities\_sci\_sm}')

f\_out.write(f'Total entities detected {en\_ner\_bc5cdr\_md}: {total\_entities\_bc5cdr\_md}')

f\_out.write(f'Top 30 entities {en\_core\_sci\_sm}: {len(tokens\_bc5cdr\_md.most\_common(30))}')

f\_out.write(f'Top 30 entities {en\_ner\_bc5cdr\_md}: {len(tokens\_sci\_sm.most\_common(30))}')

f\_out.write(f'Total extracted entities {en\_core\_sci\_sm}: {len(diseases\_model\_sci\_sm) + len(drugs\_model\_sci\_sm)}')

f\_out.write(f'Total extracted entities {en\_ner\_bc5cdr\_md}: {len(diseases\_model\_bc5cdr\_md) + len(drugs\_model\_bc5cdr\_md)}')

f\_out.write(f'All entities {en\_core\_sci\_sm}: {entities\_sci\_sm}')

f\_out.write(f'All entities {en\_ner\_bc5cdr\_md}: {entities\_bc5cdr\_md}')

# Read the text from the file in chunks

chunk\_size = 700

while True:

chunk = file.read(chunk\_size)

if not chunk:

break

doc\_biobert = nlp(chunk)

entities\_biobert += [entity['word'] for entity in doc\_biobert]

filtered\_biobert = [entity['word'] for entity in doc\_biobert if entity['entity'] == 'DRUG' or entity['entity'] == 'DISEASE']

total\_biobert += len(entities\_biobert)

tokens\_biobert += Counter(filtered\_biobert)

print(f'Total entities detected {MODEL\_NAME}: {total\_biobert}')

f\_out.write(f'Total entities detected {MODEL\_NAME}: {total\_biobert}')

f\_out.write(f'Top 30 entities {MODEL\_NAME}: {tokens\_biobert.most\_common(30)}')

f\_out.write(f'Total extracted entities {MODEL\_NAME}: {len(filtered\_biobert)}')

f\_out.write(f'All entities {MODEL\_NAME}: {entities\_biobert}')

file.close()

f\_out.close()

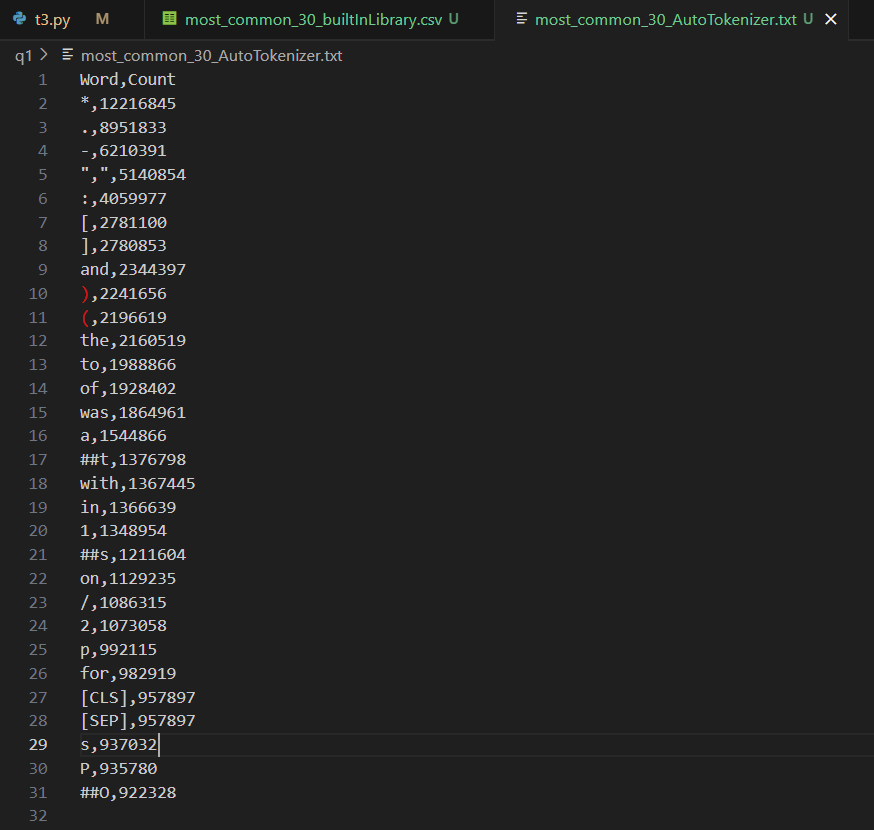
run()

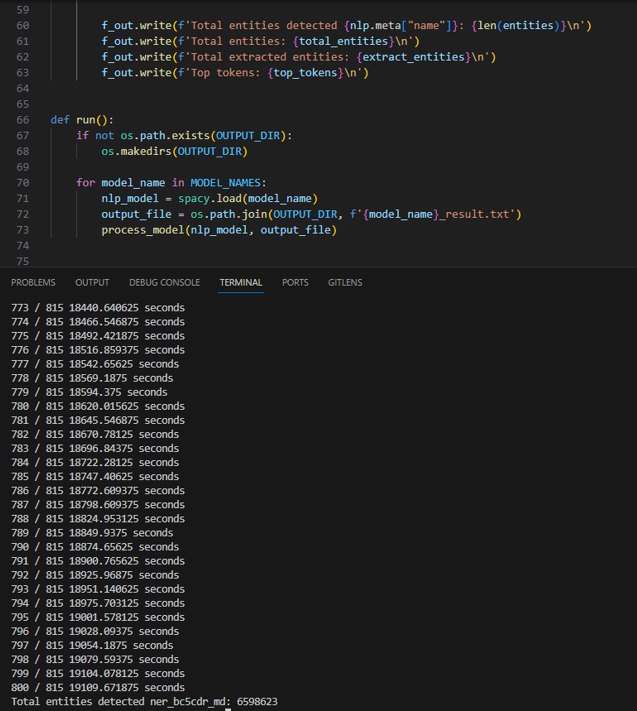
**#Output**

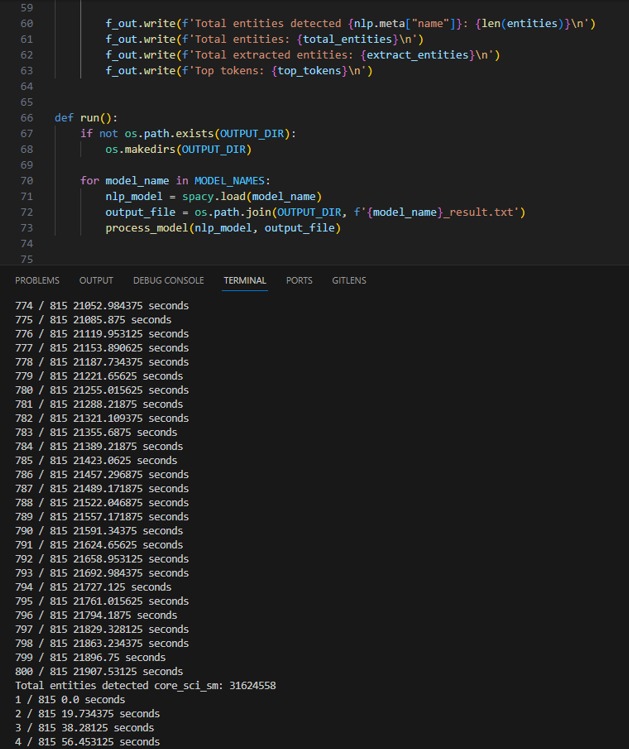
what’s the difference:

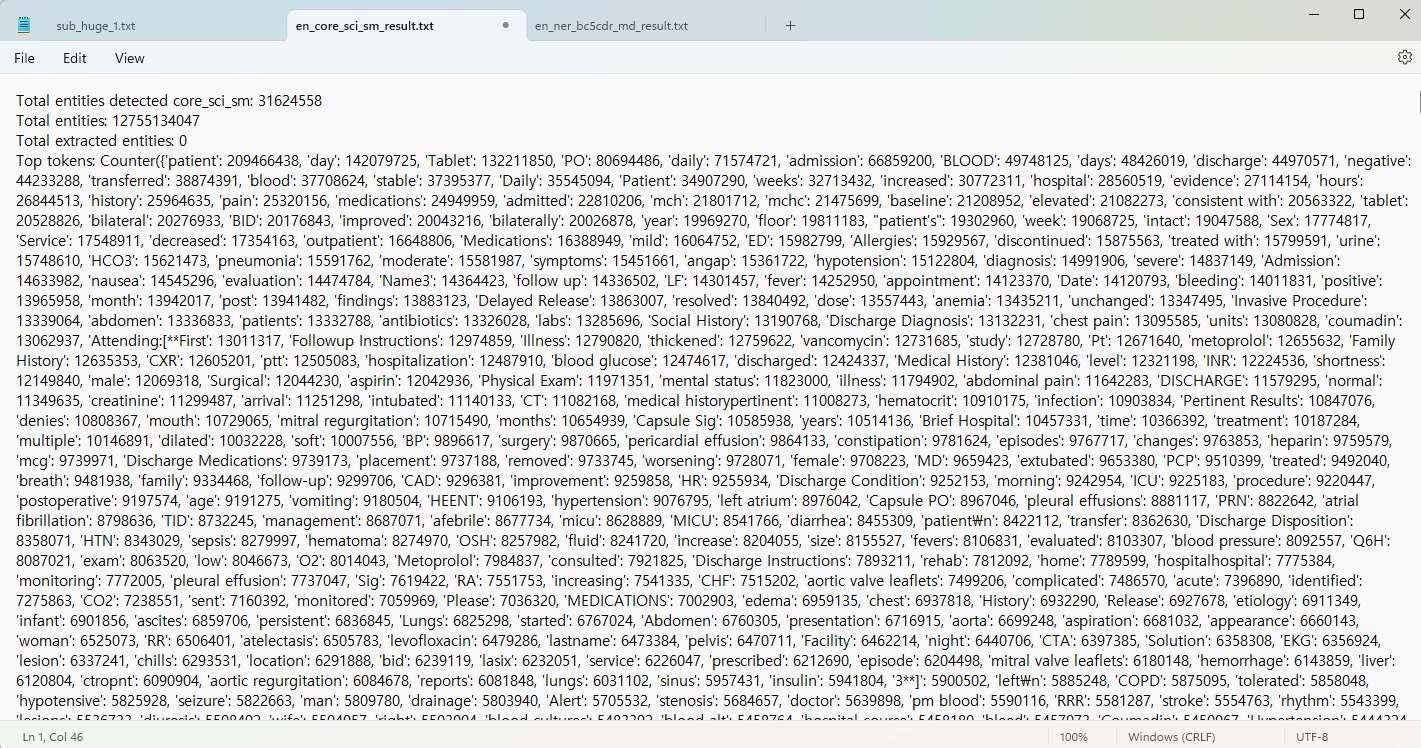
'en\_ner\_bc5cdr\_md' can label entities with diseases, drugs, and other symptoms, but the other two models only label as 'ENTITY'. So, if a user wants to filter words, they need to prepare a list of labels (e.g., dyslipidemia, hypertension as symptoms) and add logic to extract the requirements.

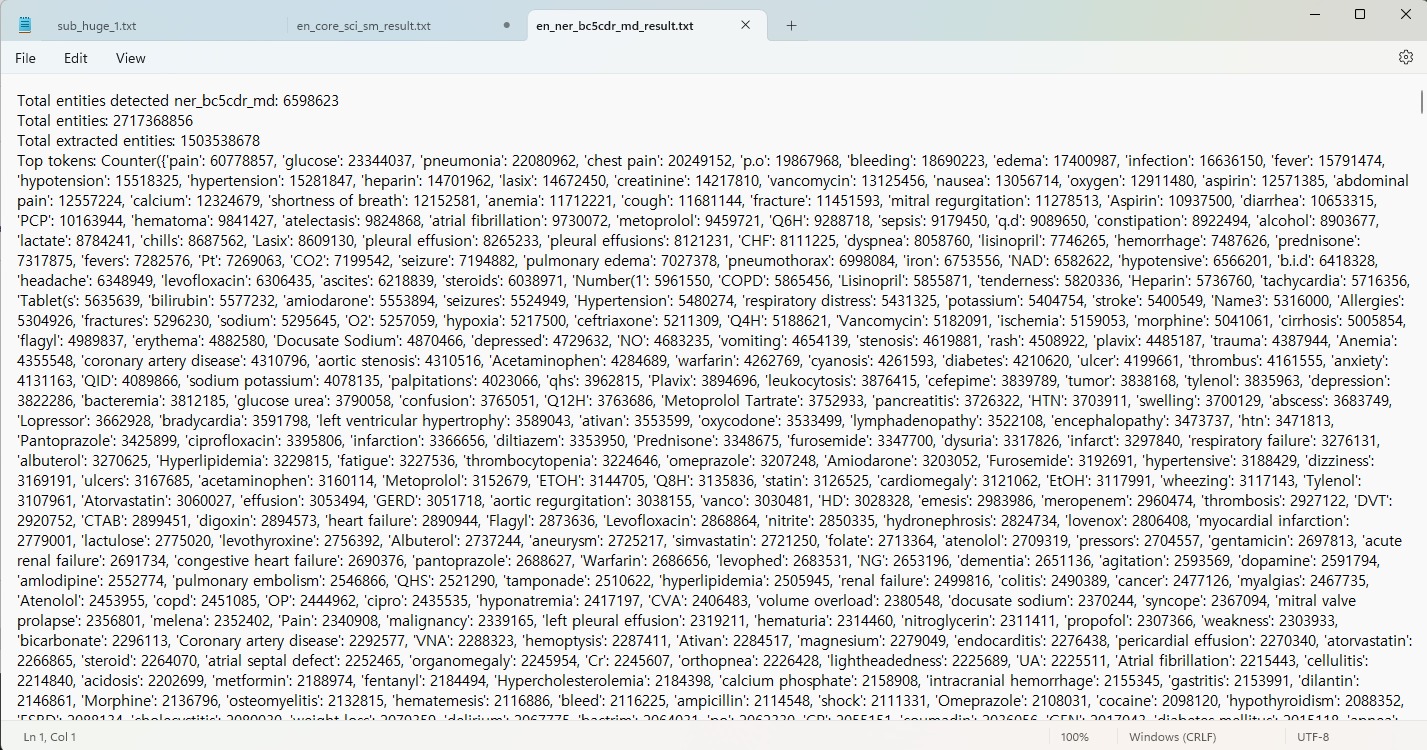
Also, BioBERT with Transformers can only handle a token count of 512 compared to a text length of 1,000,000 for NER models and the parser. Therefore, it is advisable to use a fine-tuned model to process faster and more efficiently.

Most common 30 words with BioBERT.txt









# Question 2

## Chapter 1: The Gatekeeper

**Programming code:**

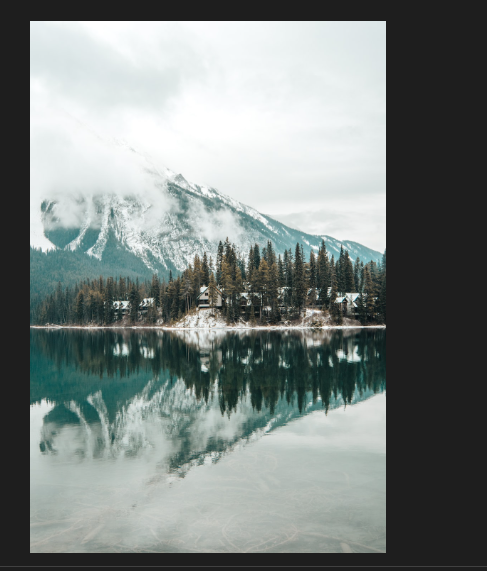
****

Figure 1: chapter1.jpg

import os, sys

from PIL import Image

import time

#Open the original picture.

image\_direction = "chapter1.jpg"

provided\_image = Image.open(image\_direction)

#Produce the figure

live\_time = int(time.time())

produced\_number = (live\_time % 100) + 50

if produced\_number % 2 == 0:

    produced\_number += 10

#Make a fresh image that is identical to the first in terms of size and mode

new\_image = Image.new("RGB", provided\_image.size)

#Get each image's pixel data.

initial\_pixels = provided\_image.load()

new\_pixels = new\_image.load()

#Iterate through every pixel, make changes, and figure out how much each red pixel is worth.

red\_sum = 0

width, height = provided\_image.size

for i in range(width):

    for j in range(height):

        r, g, b = initial\_pixels[i, j]

        altered\_pixel = (r + produced\_number, g + produced\_number, b + produced\_number)

        new\_pixels[i, j] = altered\_pixel

        red\_sum += altered\_pixel[0]  #Build up your red values.

#Save the updated picture.

output\_image\_path = "chapter1out.jpg"

new\_image.save(output\_image\_path)

#Show a notification that the procedure is finished.

print(f"Updated picture with changed pixels saved as {output\_image\_path}")

print(f"Total red pixel values in the updated picture: {red\_sum}")

**#Output**

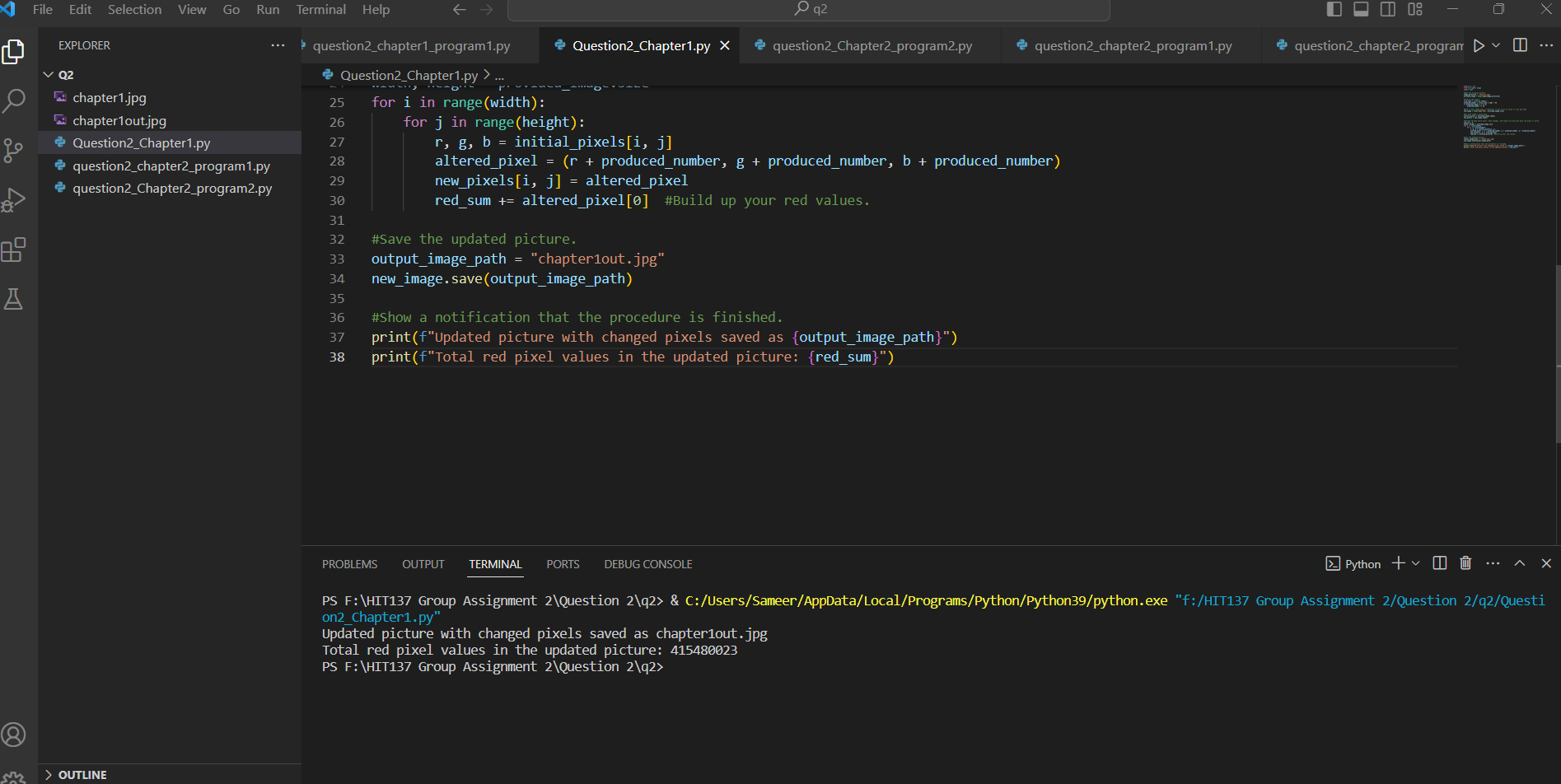
****

Figure 2: Output of Question2\_Chapter1.py

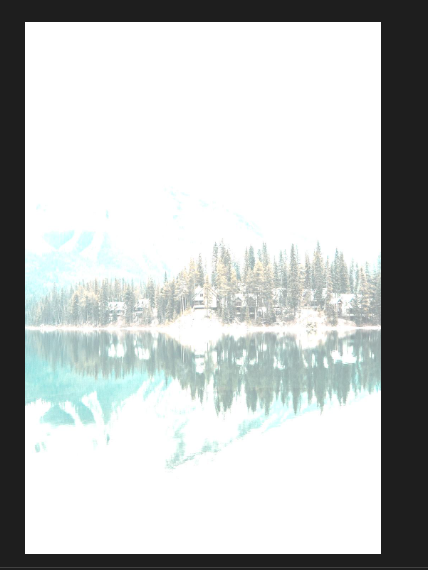
****

Figure 3: chapter1out.jpg

## Chapter 2: The Chamber of Strings

### First program:

**Programming code:**

def divide\_and\_seperate(a):

    number\_string = ''

    letter\_string = ''

    for char in a:

        if char.isdigit():

            number\_string += char

        elif char.isalpha():

            letter\_string += char

    #ASCII values of characters if digit is even

    even\_numbers\_ascii = [str(ord(char)) for char in number\_string if int(char) % 2 == 0]

    #contain the ASCII values of the uppercase letters

    upper\_case\_ascii = [str(ord(char)) for char in letter\_string if char.isupper()]

    return number\_string, letter\_string, even\_numbers\_ascii, upper\_case\_ascii

#example from question

input\_question\_string = '56aAw1984sktr235270aYmn145ss785fsq3100'

#calling divide\_and\_seperate function

result = divide\_and\_seperate(input\_question\_string)

number\_string, letter\_string, even\_numbers\_ascii, upper\_case\_ascii = result

print(f'Seperate Number String: {number\_string}')

print(f'Seperate Letter String: {letter\_string}')

print(f'Even Numbers to ASCII Code: {", ".join(even\_numbers\_ascii)}')

print(f'Upper-case Letters to ASCII Code: {", ".join(upper\_case\_ascii)}')

**Output:**

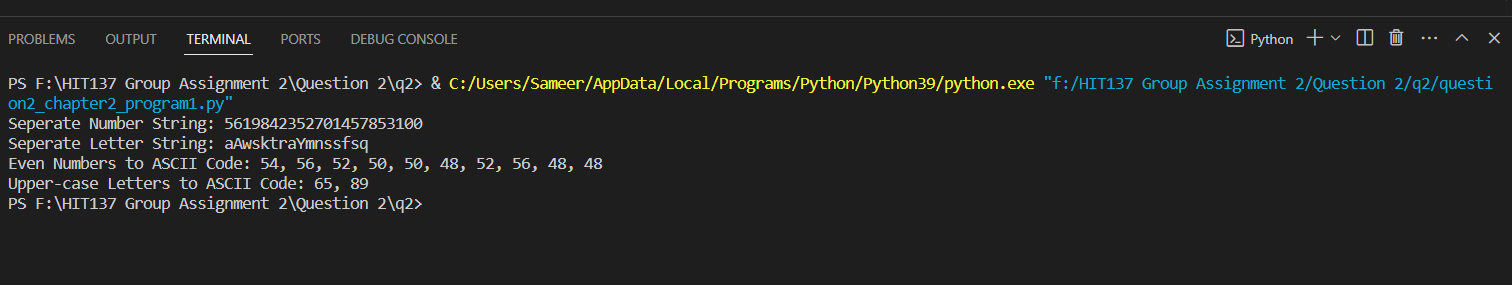
****

Figure 4: Output of question2\_chapter2\_program1.py

### Second program:

**Programming code:**

def decrypt(cryptogram, shift):

    decrypted\_text = ""

    for char in cryptogram:

        if char.isalpha():

            if char.isupper():

                decrypted\_text += chr((ord(char) - shift - 65) % 26 + 65)

            else:

                decrypted\_text += chr((ord(char) - shift - 97) % 26 + 97)

        else:

            decrypted\_text += char

    return decrypted\_text

def find\_shift\_key(cryptogram):

    for shift in range(1, 26):

        decrypted\_text = decrypt(cryptogram, shift)

        print(f"Shift Key: {shift}, Decrypted Text: {decrypted\_text}")

# Provided cryptogram

cryptogram = "VZ FRYSVFU VZCNGVIRAG NAQ N YUGGYR VAFRPHEIR V ZNXR ZVEGNXRF V NZ BHG US PRAGEBY NAQNG GVZRE UNEQ GB UNAQYIR OHG VS LBH PNAG UNAQYR ZR NG ZL JBEFG GURA LOH HER N URYYQBAG URFREIR ZR NG ZL ORFG ZNEVYLA ZEAEBR"

# Find the shift key(s)

find\_shift\_key(cryptogram)

**Output:**

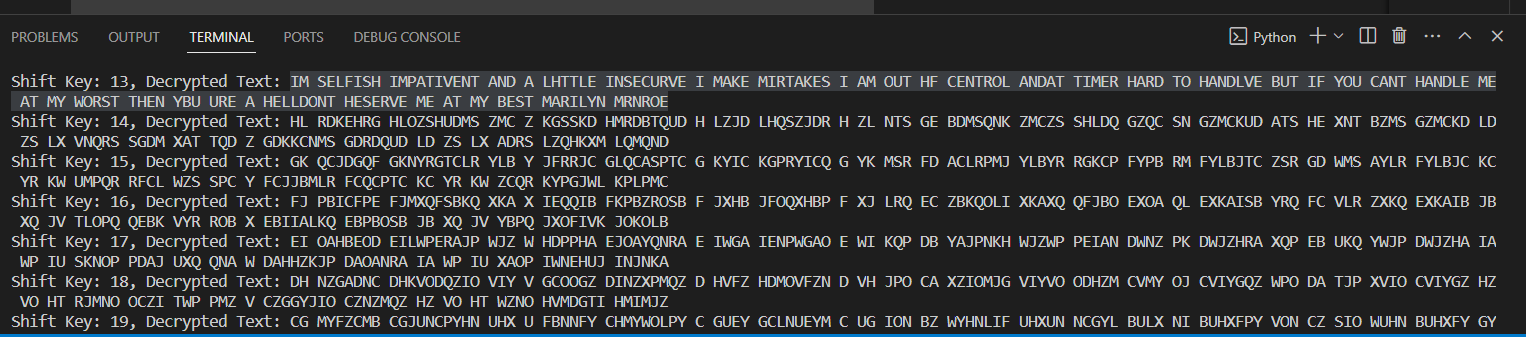
****

Figure 5: Output of question2\_Chapter2\_program2.py

# Question 3

## 1. Fixing the next code will reveal the key.

**Programming code:**

total = 0

for i in range(5):

    for j in range(3):

        if i + j == 5:

            total += i + j

        else:

            total -= i - j

counter = 0

while counter < 5:

    if total < 13:

        total += 1

    elif total > 13:

        total = 1

    else:

        counter += 2

print("Final Total:", total)

**Output:**

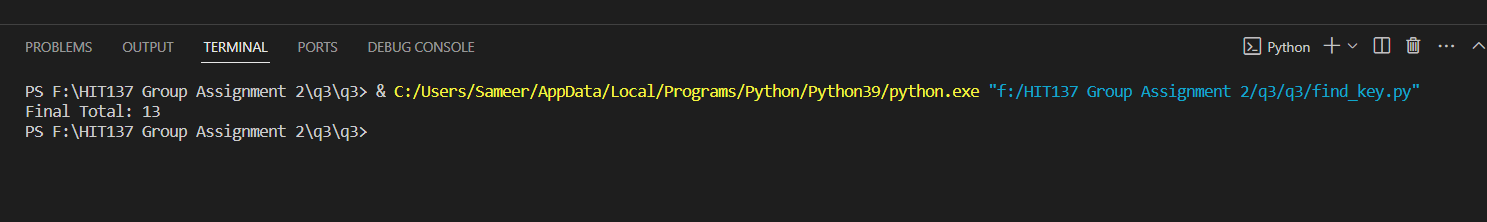
****

Figure 6:Output of find\_key.py

## 2. Write the decryption function to decrypt the ‘encrypted code’ to the original code.

**Programming code:**

#This is encrypt function which takes a plaintext and a key

def encrypt(text, key):

    encrypted\_text = ""

    for char in text: #Iterates over each character

        if char.isalpha():#If character is alphabetic

            shifted = ord(char) + key#calculates the shifted value by adding the key to the ASCII value of the character

            if char.islower():#if character is lowercase

                #if the character is beyond the lowercase range('a' to 'z')

                if shifted > ord('z'):

                    shifted -= 26

                elif shifted < ord('a'):

                    shifted += 26

            #If the character is uppercase

            elif char.isupper():

                if shifted > ord('Z'):

                    shifted -= 26

                elif shifted < ord('A'):

                    shifted += 26

            encrypted\_text += chr(shifted)

        else:

            # non-alphabetic characters unchanged

            encrypted\_text += char

    return encrypted\_text

#Function to decrypt an encrypted text

def decrypt(encrypted\_code, key):

    return encrypt(encrypted\_code, -key)

#key value result from find\_key.py program

key = 13

#Encrypted code

encrypted\_code = """

tybony\_inevnoyr = 100

zl\_qvpg = {'xrl1': 'inyhr1', 'xrl2': 'inyhr2', 'xrl3': 'inyhr3'}

qrs cebprff\_ahzoref():

    tybony tybony\_inevnoyr

    ybpny\_inevnoyr = 5

    ahzoref = [1, 2, 3, 4, 5]

    juvyr ybpny\_inevnoyr > 0:

        vs ybpny\_inevnoyr % 2 == 0:

            ahzoref.erzbir(ybpny\_inevnoyr)

        ybpny\_inevnoyr -= 1

    erghea ahzoref

zl\_frg = {1, 2, 3, 4, 5, 5, 4, 3, 2, 1}

erfhyg = cebprff\_ahzoref(ahzoref=zl\_frg)

qrs zbqvsl\_qvpg():

    ybpny\_inevnoyr = 10

    zl\_qvpg['xrl4'] = ybpny\_inevnoyr

zbqvsl\_qvpg(5)

qrs hcongr\_tybony():

    tybony tybony\_inevnoyr

    tybony\_inevnoyr += 10

sbe v va enatr(5):

    cevag(v)

    v += 1

vs zl\_frg vf abg Abar naq zl\_qvpg['xrl4'] == 10:

    cevag("Pbaqvgvba zrg!")

vs 5 abg va zl\_qvpg:

    cevag("5 abg sbhaq va gur qvpgvbanel!")

cevag(tybony\_inevnoyr)

cevag(zl\_qvpg)

cevag(zl\_frg)

"""

# call decrypt function with value of encrypted\_code and key

decrypted\_code = decrypt(encrypted\_code, key)

# Result of decrypted code

print(decrypted\_code)

**Output:**

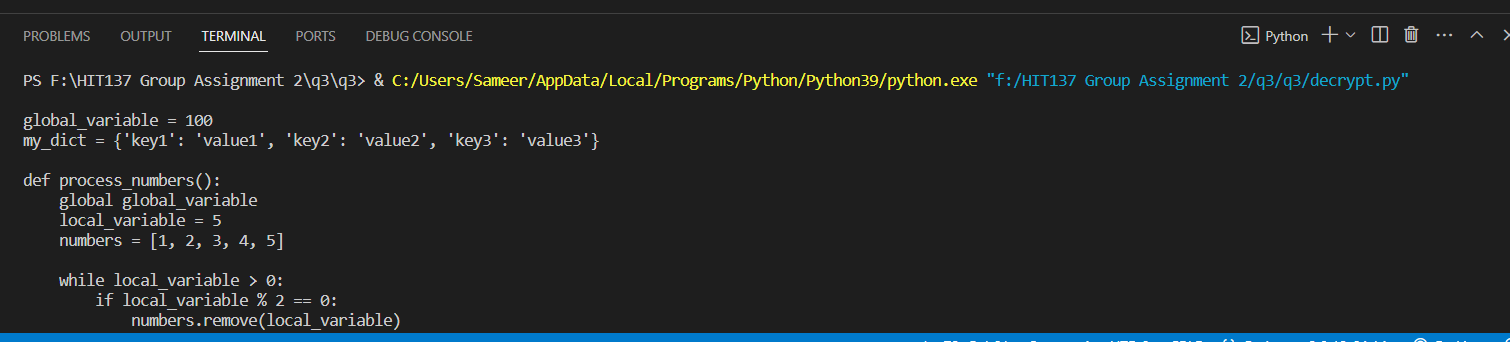
****

Figure 7: Output of decrypt.py

## 3. Correct the errors and provide the comments

**Programming Code:**

global\_variable = 100 #global\_variable is set to 100

my\_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'} # my\_dic is initialized with some key-values

def process\_numbers(numbers): #function process\_numbers take parameter of numbers

    global global\_variable

    local\_variable = 5

    numbers = [1, 2, 3, 4, 5] #local variable numbers is initialized to [,2,3,4,5]

    while local\_variable > 0: #while loop is used to iterate local varible is greater than 0

        if local\_variable % 2 == 0: #if it is even numbers

            numbers.remove(local\_variable) #removes number from list

        local\_variable -= 1

    return numbers #returns modified numbers

my\_set = {1, 2, 3, 4, 5, 5, 4, 3, 2, 1} #my\_set is inialized

result = process\_numbers(numbers=my\_set)

def modify\_dict(local\_variable ): # function modify\_dict takes parameter local\_variable

    my\_dict['key4'] = local\_variable #Adds a new key to my\_dict whith the value of local\_variable

modify\_dict(10)

def upbate\_global():

    global global\_variable

    global\_variable += 10 #increaments by 10

for i in range(5):#Iterates over the range 0 to 4

    print(i)

    i += 1

if my\_set is not None and my\_dict['key4'] == 10:

    print("Condition met!")

if 5 not in my\_dict:

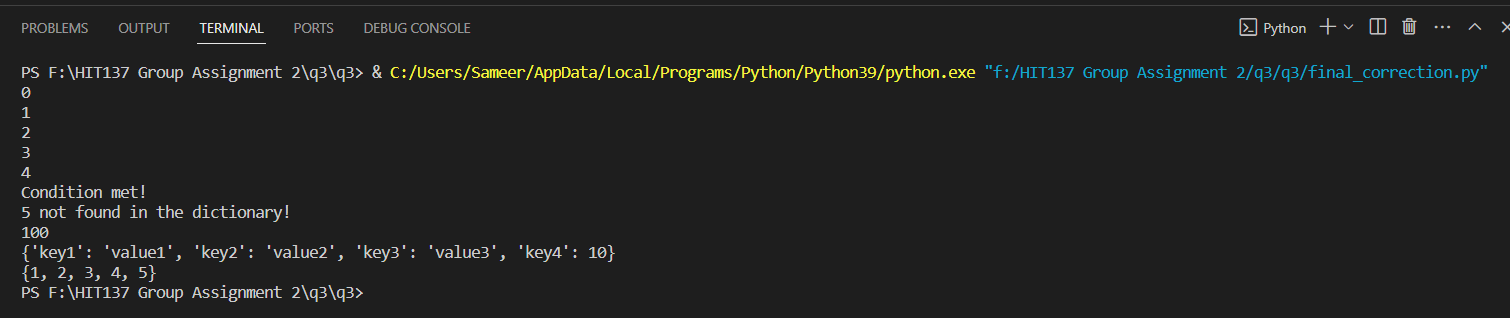
    print("5 not found in the dictionary!")

print(global\_variable) #print value of global\_variable

print(my\_dict)#print value of my\_dict

print(my\_set)#print value of my\_set

**Output:**

****

# Question 4

Welcome to the final task of this assignment. You are required to create a GitHub repository and add all your group mates to it (make sure to keep it public, not private). You should do this before you start the assignment. All the answers and contributions should be recorded in GitHub till you submit the assignment

**Output:**

<https://github.com/Sameer84/HIT137-Assignment-02.git>

**Group Members- Github profile**  
[Sameer Basnet] - [Sameer84]  
[Samir Dhakal] - [srd199]  
[Susanti Djie] - [SusantiDj]  
[Nishat Anjum] - [Nishat199]

