HIT137 Software Now

Assignment2

Submitted By:

Naresh Kumar Chaudhary [s371375]

Shailendra Adhikari [s346551]

Devansh Sharma [s357310]

Sajjad Hossain [s375240]

Github Repository:

<https://github.com/S371375/Group-106-DAR---HIT137-SOFTWARE-NOW>

Table of Contents

[Question 1: Use the open-source NLP(Natural Language Processing) libraries and perform various tasks. 3](#_Toc156745082)

[Task 1: Extract the ‘text’ in all the CSV files and store them into a single ‘.txt file’. 3](#_Toc156745083)

[Task 2:Research 4](#_Toc156745084)

[Task3:Programming and Research 6](#_Toc156745085)

[Task3.1: Using any in-built library present in Python, count the occurrences of the words 6](#_Toc156745086)

[Task3.2: 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. 7](#_Toc156745087)

[Task 4: Named-Entity Recognition (NER) 8](#_Toc156745088)

[Question2 11](#_Toc156745089)

[Chapter1:The Gatekeeper 11](#_Toc156745090)

[Chapter 2: The Chamber of Strings 13](#_Toc156745091)

[Assume s is a string 13](#_Toc156745092)

[Find the shift key (s) the gives the original quote. 13](#_Toc156745093)

[Question3: Fixing the error-prone codes. 14](#_Toc156745094)

[Finding the key 14](#_Toc156745095)

[Write the decryption function to decrypt the ‘encrypted code’ to the original code. 15](#_Toc156745096)

[Correct the errors and provide the comments. 16](#_Toc156745097)

[Question 4: create a GitHub repository and add all your group mates to it (make sure to keep it public, not private). 17](#_Toc156745098)

# Question 1: Use the open-source NLP(Natural Language Processing) libraries and perform various tasks.

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

Created the virtual environment

python3 -m venv assignment2

source assignment2/bin/activate

Installed the library pandas

pip install pandas

import os

import pandas as pd

from zipfile import ZipFile

# Specify the path to the zipped folder

zipped\_folder\_path = 'Assignment 2.zip'

extracted\_folder\_path = 'extracted\_files'

# Extract the contents of the zipped folder

with ZipFile(zipped\_folder\_path, 'r') as zip\_ref:

zip\_ref.extractall(extracted\_folder\_path)

# List all CSV files in the extracted folder

csv\_files = [file for file in os.listdir(extracted\_folder\_path) if file.endswith('.csv')]

# Create an empty list to store text from all CSV files

all\_texts = []

# Iterate through each CSV file

for csv\_file in csv\_files:

# Read the CSV file

df = pd.read\_csv(os.path.join(extracted\_folder\_path, csv\_file))

# Try 'TEXT' column, if not present, try 'SHORT-TEXT' column

text\_column\_name = 'TEXT' if 'TEXT' in df.columns else 'SHORT-TEXT'

# Concatenate text from the specified column

all\_texts.extend(df[text\_column\_name].astype(str).tolist())

# Combine all texts into a single string

combined\_text = '\n'.join(all\_texts)

# Write the combined text to a single .txt file

output\_txt\_path = 'output/Q1-task1\_output.txt'

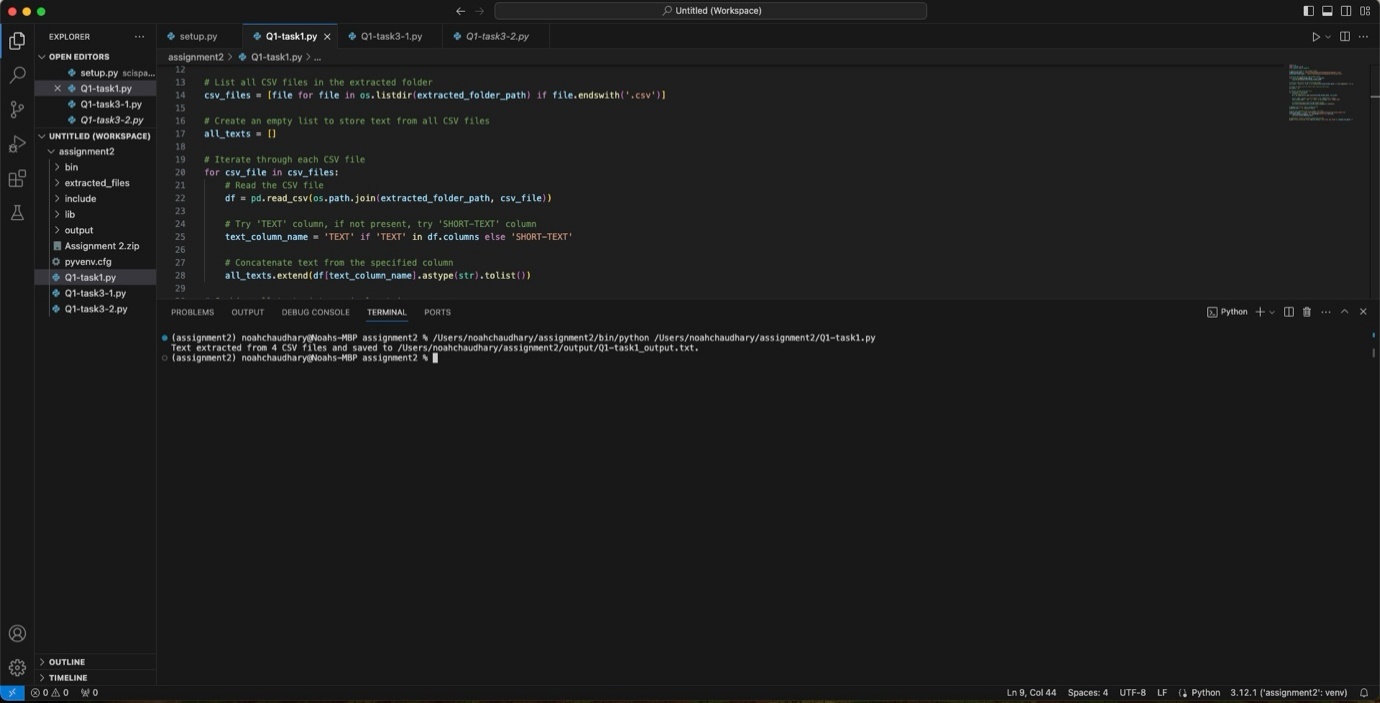
with open(output\_txt\_path, 'w', encoding='utf-8') as txt\_file:

txt\_file.write(combined\_text)

# Optionally, print a message indicating success

print(f"Text extracted from {len(csv\_files)} CSV files and saved to {output\_txt\_path}.")

Output:



## Task 2:Research

Install the libraries(SpaCy – scispaCy – ‘en\_core\_sci\_sm’/’en\_ner\_bc5cdr\_md’).

Install the libraries (Transformers (Hugging Face) - and any bio-medical model

(BioBert) that can detect drugs, diseases, etc from the text).

Installed libraries:

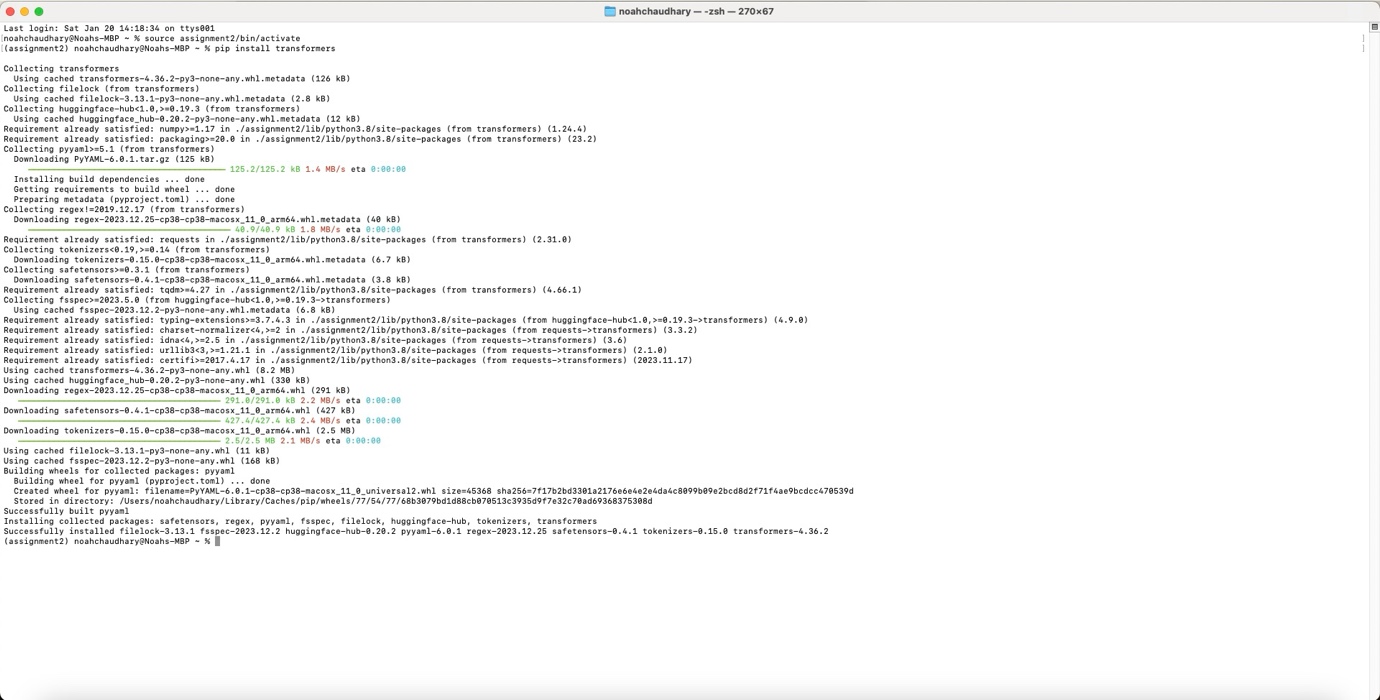
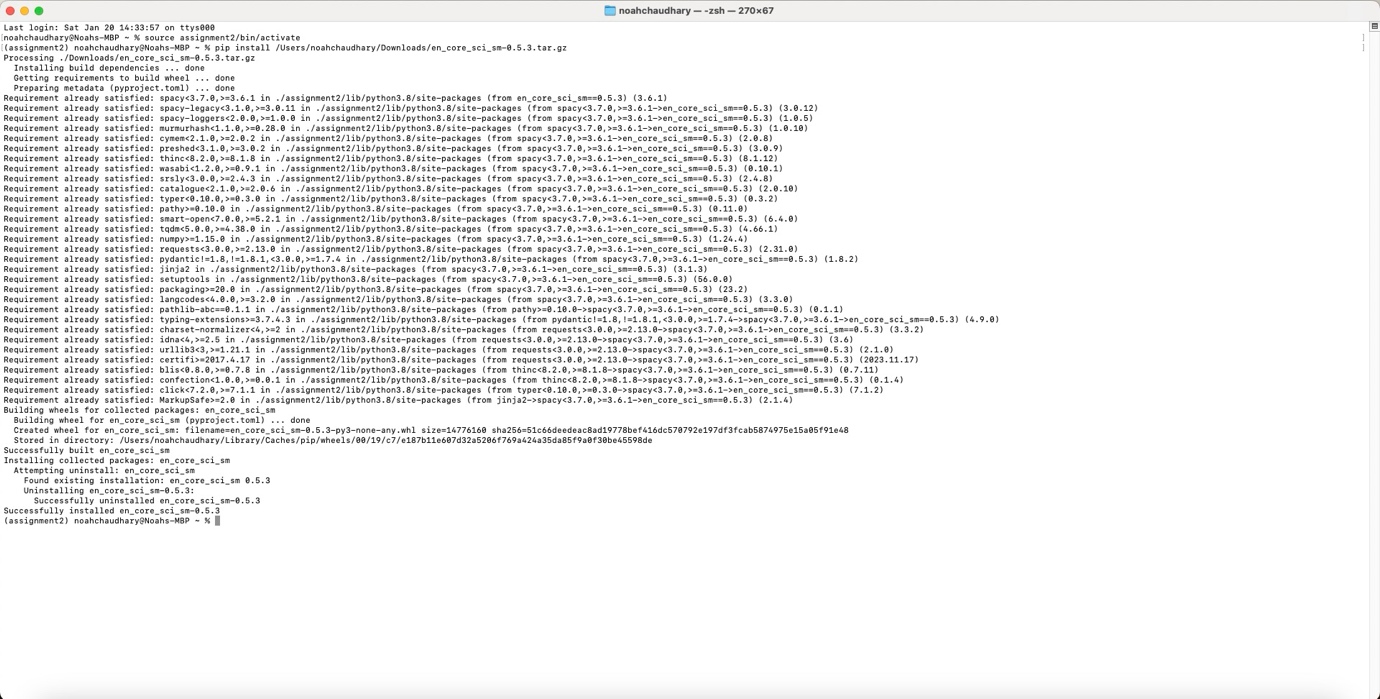
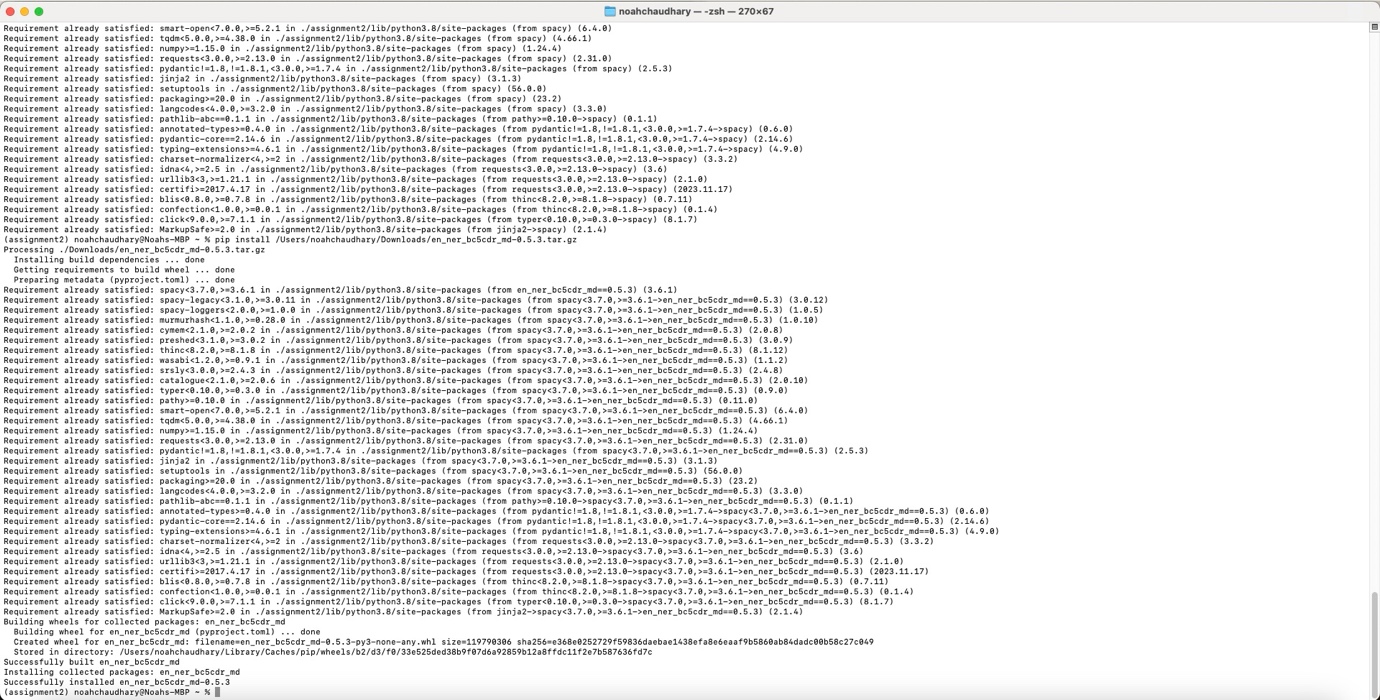
pip install spacy

pip install en\_core\_sci\_sm-0.5.3.tar.gz

ppip install en\_ner\_bc5cdr\_md-0.5.3.tar.gz

pip install transformers

Output:



## Task3:Programming and Research

### Task3.1: 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.

import csv

from collections import Counter

import string

# Read the text from the .txt file

with open('/Users/noahchaudhary/assignment2/output/Q1-task1\_output.txt', 'r', encoding='utf-8') as txt\_file:

text = txt\_file.read()

# Remove punctuation and convert to lowercase

translator = str.maketrans('', '', string.punctuation)

text = text.translate(translator).lower()

# Tokenize the text into words

words = text.split()

# Count word occurrences

word\_counts = Counter(words)

# Get the top 30 most common words

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

# Store the top 30 words and their counts in a CSV file

csv\_file\_path = 'output/Q1-task3-1\_output.csv'

with open(csv\_file\_path, 'w', newline='', encoding='utf-8') as csv\_file:

csv\_writer = csv.writer(csv\_file)

# Write header

csv\_writer.writerow(['Word', 'Count'])

# Write data

csv\_writer.writerows(top\_30\_words)

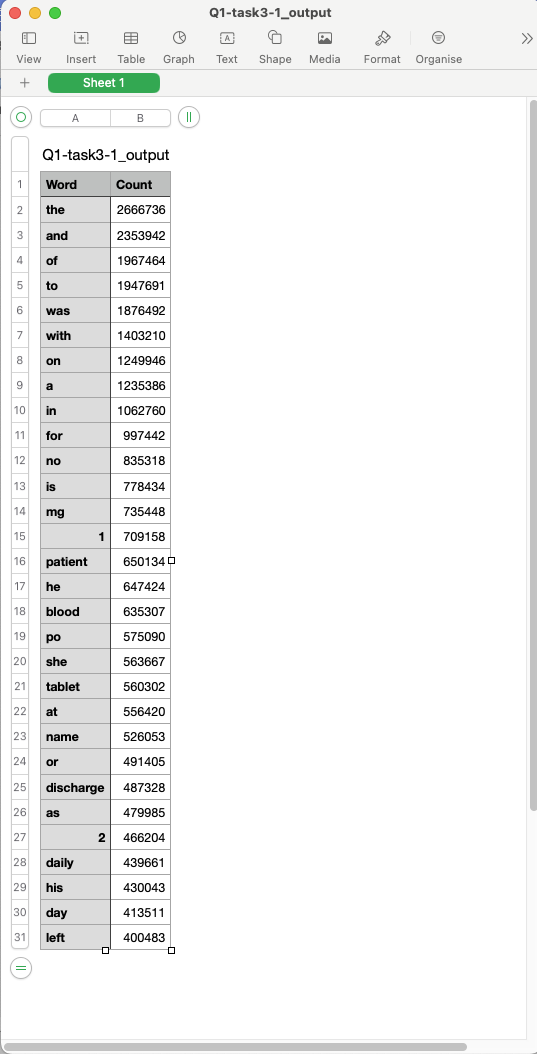
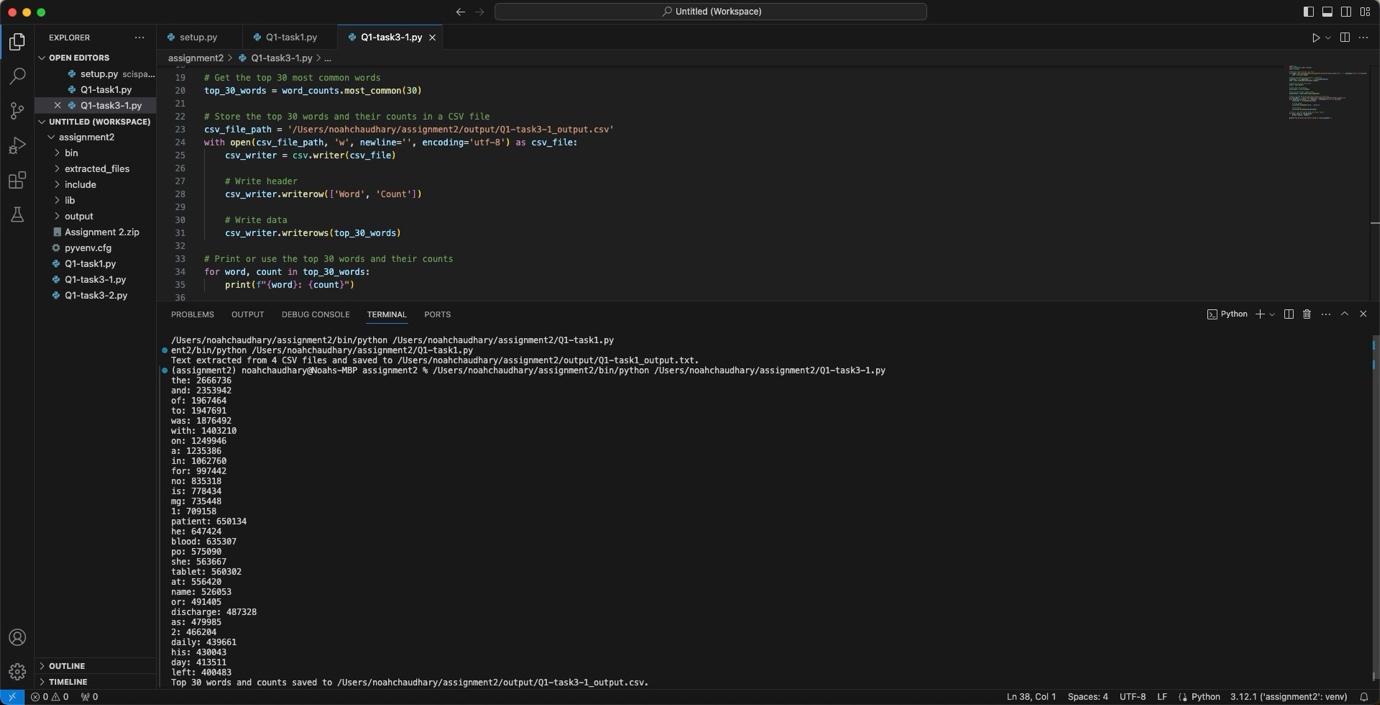
# Print or use the top 30 words and their counts

for word, count in top\_30\_words:

print(f"{word}: {count}")

print(f"Top 30 words and counts saved to {csv\_file\_path}.")

Output:



### Task3.2: 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.

from transformers import AutoTokenizer

from collections import Counter

def read\_text\_file(file\_path):

"""Read the content of a text file."""

try:

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

return file.read()

except FileNotFoundError:

print(f"Error: File not found - {file\_path}")

return None

except Exception as e:

print(f"Error reading file: {e}")

return None

def count\_unique\_tokens(text, tokenizer, top\_n=30):

"""Count the occurrences of unique tokens in the text."""

try:

# Tokenize the text

tokens = tokenizer.tokenize(text)

# Count token occurrences

token\_counts = Counter(tokens)

# Get the top N tokens

top\_tokens = token\_counts.most\_common(top\_n)

return top\_tokens

except Exception as e:

print(f"Error counting tokens: {e}")

return None

def main():

# File path to the text file

text\_file\_path = 'output/Q1-task1\_output.txt'

# Load the BioBERT tokenizer

tokenizer = AutoTokenizer.from\_pretrained("dmis-lab/biobert-base-cased-v1.2")

# Read the text file

text\_content = read\_text\_file(text\_file\_path)

if text\_content is not None:

# Count and retrieve top tokens

top\_tokens = count\_unique\_tokens(text\_content, tokenizer, top\_n=30)

if top\_tokens is not None:

# Display the top tokens and their counts

for token, count in top\_tokens:

print(f"Token: {token}, Count: {count}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## 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.)

import spacy

from transformers import AutoTokenizer, AutoModelForTokenClassification

import torch

from collections import Counter

# Suppress FutureWarning from spaCy

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

# Load spaCy models

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

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

# Load BioBERT model and tokenizer

biobert\_tokenizer = AutoTokenizer.from\_pretrained("monologg/biobert\_v1.1\_pubmed")

biobert\_model = AutoModelForTokenClassification.from\_pretrained("monologg/biobert\_v1.1\_pubmed")

# Read the content of the text file

file\_path = 'output/Q1-task1\_output.txt'

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

text = file.read()

# Process the text using spaCy models

doc\_sci\_sm = nlp\_sci\_sm(text[:1000000]) # Process only the first 1,000,000 characters

doc\_bc5cdr\_md = nlp\_bc5cdr\_md(text[:1000000]) # Process only the first 1,000,000 characters

# Process the text using BioBERT

tokens = biobert\_tokenizer(text[:512], return\_tensors='pt') # Process only the first 512 tokens for BioBERT

with torch.no\_grad():

outputs = biobert\_model(\*\*tokens)

# Extract 'diseases' and 'drugs' entities separately

def extract\_entities(doc, label):

return [ent.text for ent in doc.ents if ent.label\_ == label]

diseases\_sci\_sm = extract\_entities(doc\_sci\_sm, 'DISEASE')

drugs\_sci\_sm = extract\_entities(doc\_sci\_sm, 'CHEMICAL')

diseases\_bc5cdr\_md = extract\_entities(doc\_bc5cdr\_md, 'DISEASE')

drugs\_bc5cdr\_md = extract\_entities(doc\_bc5cdr\_md, 'CHEMICAL')

# Extract entities from BioBERT outputs

bio\_entities = biobert\_tokenizer.convert\_ids\_to\_tokens(torch.argmax(outputs.logits, dim=2).squeeze().tolist())

bio\_entities = [token for token, label in zip(bio\_entities, outputs.logits.argmax(dim=2).squeeze().tolist()) if label != 0]

# Separate BioBERT entities into 'diseases' and 'drugs'

bio\_diseases = [entity.replace("##", "") for entity in bio\_entities if entity.startswith("B-Disease") or entity.startswith("I-Disease")]

bio\_drugs = [entity.replace("##", "") for entity in bio\_entities if entity.startswith("B-Chemical") or entity.startswith("I-Chemical")]

# Compare the differences

total\_entities\_sci\_sm = len(diseases\_sci\_sm) + len(drugs\_sci\_sm)

total\_entities\_bc5cdr\_md = len(diseases\_bc5cdr\_md) + len(drugs\_bc5cdr\_md)

total\_entities\_biobert = len(bio\_diseases) + len(bio\_drugs)

common\_diseases = set(diseases\_sci\_sm) & set(diseases\_bc5cdr\_md) & set(bio\_diseases)

common\_drugs = set(drugs\_sci\_sm) & set(drugs\_bc5cdr\_md) & set(bio\_drugs)

difference\_diseases\_sci\_sm\_bc5cdr\_md = set(diseases\_sci\_sm) - set(diseases\_bc5cdr\_md)

difference\_drugs\_sci\_sm\_bc5cdr\_md = set(drugs\_sci\_sm) - set(drugs\_bc5cdr\_md)

difference\_diseases\_biobert = set(bio\_diseases) - set(diseases\_sci\_sm) - set(diseases\_bc5cdr\_md)

difference\_drugs\_biobert = set(bio\_drugs) - set(drugs\_sci\_sm) - set(drugs\_bc5cdr\_md)

# Print the results

print(f"Total entities (en\_core\_sci\_sm): {len(diseases\_sci\_sm)} diseases, {len(drugs\_sci\_sm)} drugs")

print(f"Total entities (en\_ner\_bc5cdr\_md): {len(diseases\_bc5cdr\_md)} diseases, {len(drugs\_bc5cdr\_md)} drugs")

print(f"Total entities (BioBERT): {len(bio\_diseases)} diseases, {len(bio\_drugs)} drugs")

print(f"Common diseases: {common\_diseases}")

print(f"Common drugs: {common\_drugs}")

print(f"Difference in diseases (en\_core\_sci\_sm vs. en\_ner\_bc5cdr\_md): {difference\_diseases\_sci\_sm\_bc5cdr\_md}")

print(f"Difference in drugs (en\_core\_sci\_sm vs. en\_ner\_bc5cdr\_md): {difference\_drugs\_sci\_sm\_bc5cdr\_md}")

print(f"Difference in diseases (BioBERT): {difference\_diseases\_biobert}")

print(f"Difference in drugs (BioBERT): {difference\_drugs\_biobert}")

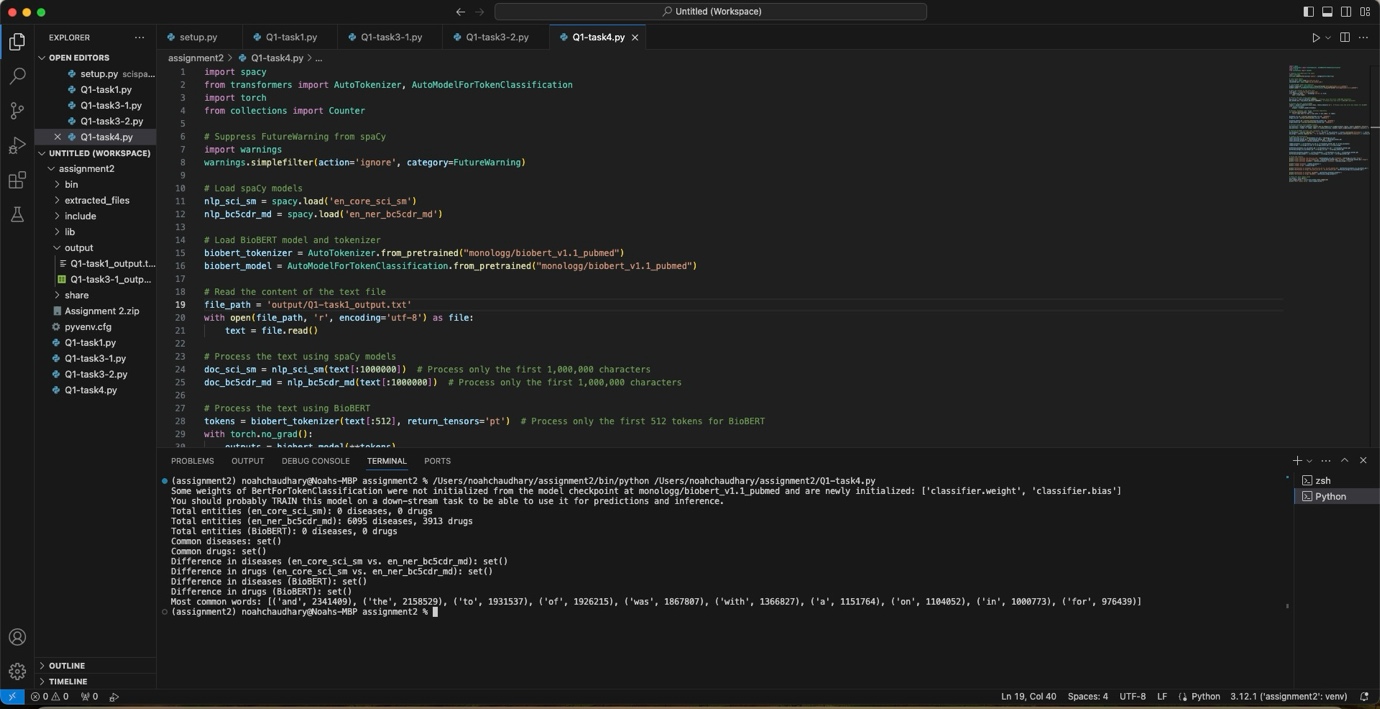
# Check for most common words

all\_words = text.split()

most\_common\_words = Counter(all\_words).most\_common(10)

print(f"Most common words: {most\_common\_words}")

Output:



# Question2

## Chapter1:The Gatekeeper

Generate a new image with the converted pixels (upload it as

‘chapter1out.png’).

Finally, add all the red (r) pixel values in the new\_image and provide the sum

as output to move to the next chapter.

from PIL import Image

# Load the image

image\_path = 'extracted\_files/chapter1.jpg'

original\_image = Image.open(image\_path)

width, height = original\_image.size

# Generate a number based on the algorithm

import time

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

generated\_number = (current\_time % 100) + 50

if generated\_number % 2 == 0:

generated\_number += 10

print(generated\_number)

# Create a new image with converted pixels

new\_image = Image.new('RGB', (width, height))

# Iterate over each pixel in the original image

for y in range(height):

for x in range(width):

# Get the original pixel values

r, g, b = original\_image.getpixel((x, y))

# Modify the pixel values based on the generated number

r\_new = (r + generated\_number) % 256

g\_new = (g + generated\_number) % 256

b\_new = (b + generated\_number) % 256

# Set the modified pixel values in the new image

new\_image.putpixel((x, y), (r\_new, g\_new, b\_new))

# Save the new image

new\_image\_path = 'output/chapter1out.png'

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

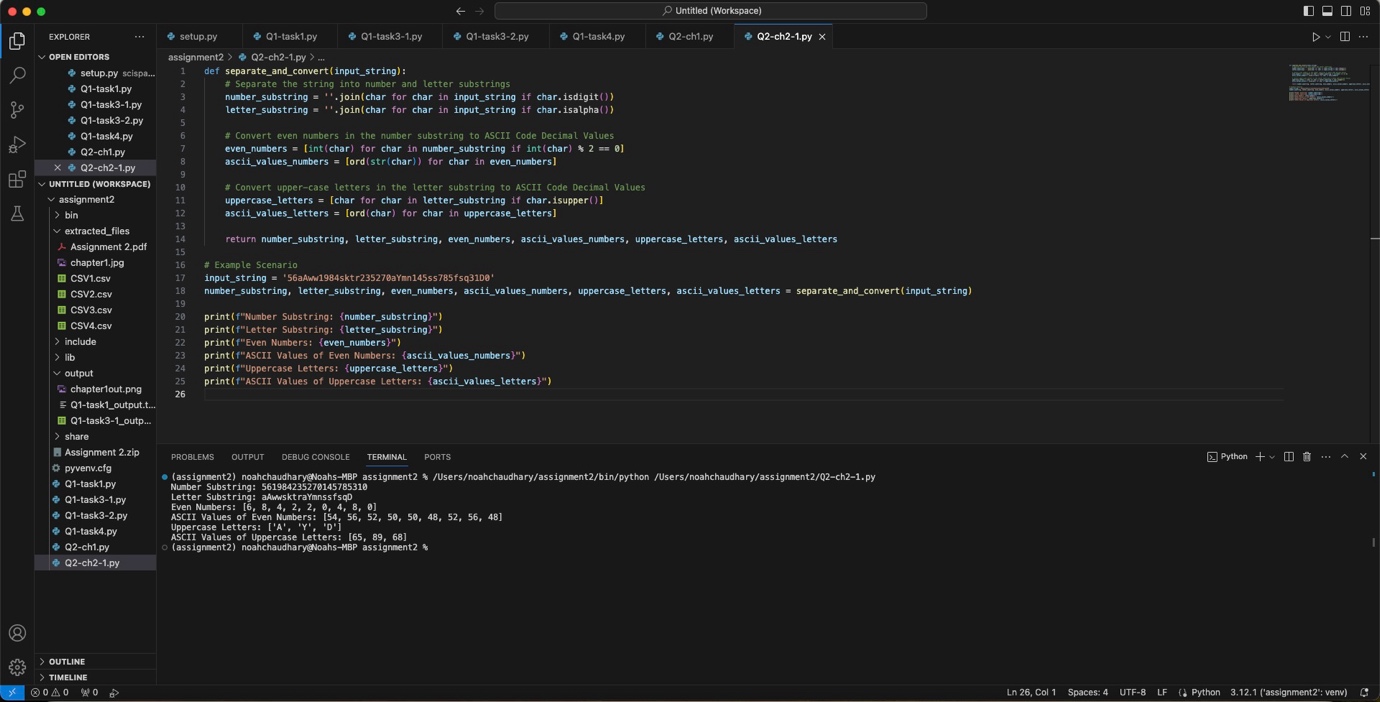
# Calculate the sum of red pixel values in the new image

red\_sum = sum([new\_image.getpixel((x, y))[0] for y in range(height) for x in range(width)])

# Print the sum for the next chapter

print(f"Sum of red pixel values: {red\_sum}")

Output:





## Chapter 2: The Chamber of Strings

### Assume s is a string

Write a program that separates a long string (at least length of 16) that contains both numbers and letters (upper and lower case) into two substrings of numbers and letters.

And then convert the even numbers in the 'number substring' and upper-case letter in the 'letter string' into ASCII Code Decimal Values.

def separate\_and\_convert(input\_string):

# Separate the string into number and letter substrings

number\_substring = ''.join(char for char in input\_string if char.isdigit())

letter\_substring = ''.join(char for char in input\_string if char.isalpha())

# Convert even numbers in the number substring to ASCII Code Decimal Values

even\_numbers = [int(char) for char in number\_substring if int(char) % 2 == 0]

ascii\_values\_numbers = [ord(str(char)) for char in even\_numbers]

# Convert upper-case letters in the letter substring to ASCII Code Decimal Values

uppercase\_letters = [char for char in letter\_substring if char.isupper()]

ascii\_values\_letters = [ord(char) for char in uppercase\_letters]

return number\_substring, letter\_substring, even\_numbers, ascii\_values\_numbers, uppercase\_letters, ascii\_values\_letters

# Example Scenario

input\_string = '56aAww1984sktr235270aYmn145ss785fsq31D0'

number\_substring, letter\_substring, even\_numbers, ascii\_values\_numbers, uppercase\_letters, ascii\_values\_letters = separate\_and\_convert(input\_string)

print(f"Number Substring: {number\_substring}")

print(f"Letter Substring: {letter\_substring}")

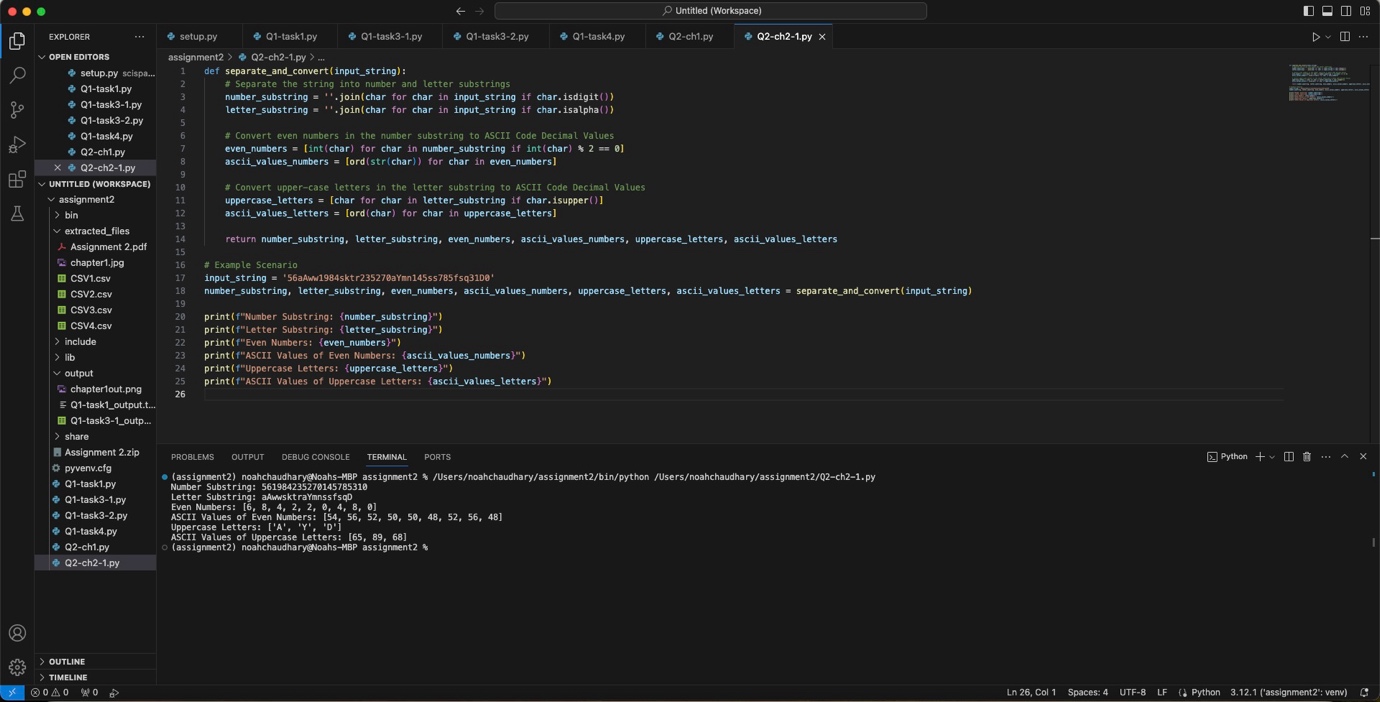
print(f"Even Numbers: {even\_numbers}")

print(f"ASCII Values of Even Numbers: {ascii\_values\_numbers}")

print(f"Uppercase Letters: {uppercase\_letters}")

print(f"ASCII Values of Uppercase Letters: {ascii\_values\_letters}")

Output:



create a program that showcases the required output for the following question:

Similarly decrypting the provided cryptogram using a 'certain' shift key value (s) gives original quote.

### Find the shift key (s) the gives the original quote.

def decrypt(text, offset=0):

decrypted = ""

for char in text:

if char.isalpha():

base\_ord = ord('A') if char.isupper() else ord('a')

decrypted += chr((ord(char) + offset - base\_ord) % 26 + base\_ord)

else:

decrypted += char

return decrypted

encrypted\_text = """VZ FRYSVFU VZCNGVRAG NAQ N YVGGYR VAFRPHER V ZNXR V NZ BHG BS PBAGEBY

NAQNG GVZRF UNEQ GB UNAQYR OHG VS LBH PNAG UNAQYR ZR NG ZL JBEFG GURA LBH FHER NF

URYYQBAG QRFREIR ZR NG ZL ORFG ZNEVYLA ZBAEBR"""

print("Encrypted text:")

print(encrypted\_text)

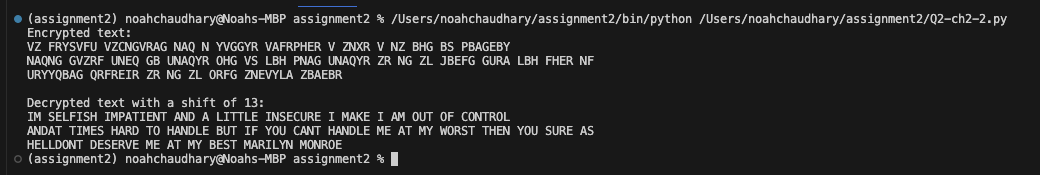
# Decrypt with a shift of 13

decrypted\_text = decrypt(encrypted\_text, 13)

print("\nDecrypted text with a shift of 13:")

print(decrypted\_text)

Output:



# Question3: Fixing the error-prone codes.

## Finding the key

total = 0

for i in range(5):

for j in range(3):

if i+j== 5:

total += 1+ j

else:

total -= 1 - j

counter = 0

while counter < 5:

if total < 13:

total += 1

elif total > 13:

total -= 1

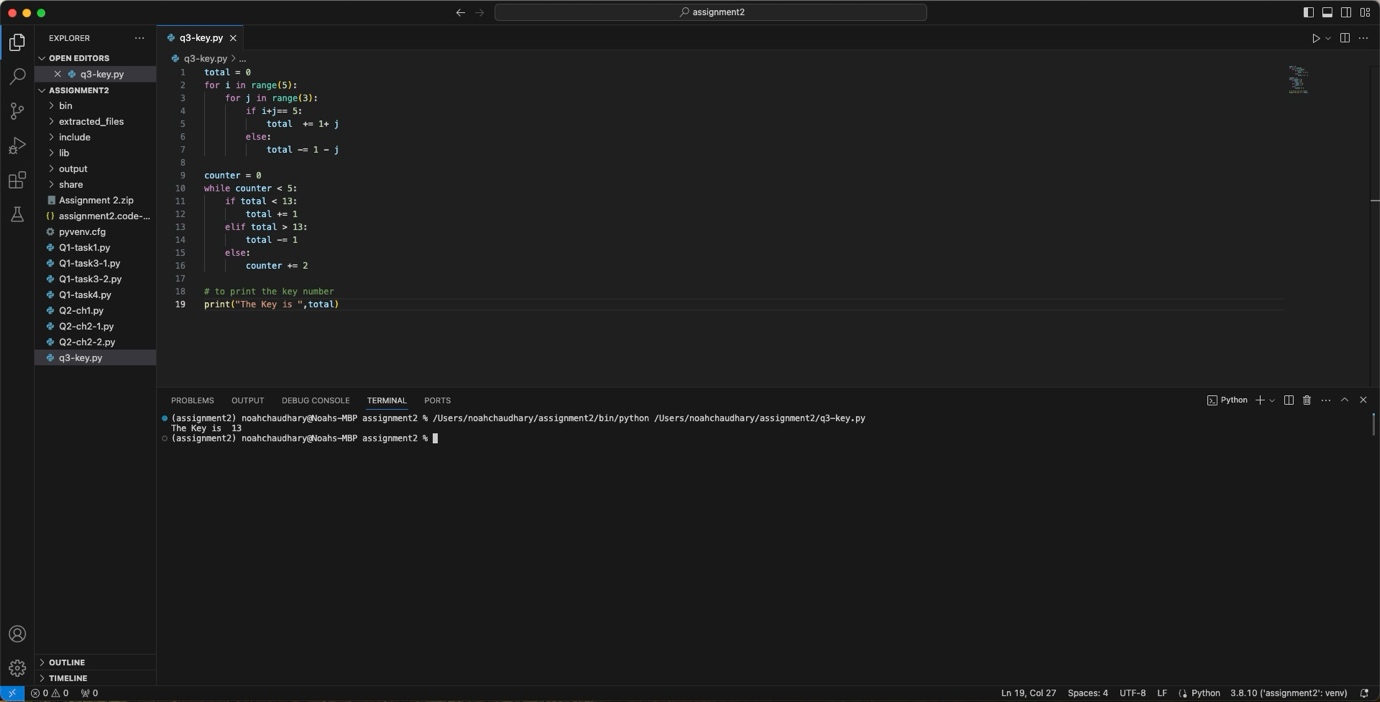
else:

counter += 2

# to print the key number

print("The Key is ",total)

Output:



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

def decrypt(text, key):

decrypted\_text = ""

for char in text:

if char.isalpha():

shifted = ord(char) - key

if char.islower():

if shifted > ord('z'):

shifted -= 26

elif shifted < ord('a'):

shifted += 26

elif char.isupper():

if shifted > ord('Z'):

shifted -= 26

elif shifted < ord('A'):

shifted += 26

decrypted\_text += chr(shifted)

else:

decrypted\_text += char

return decrypted\_text

# Given encrypted code

encrypted\_code = "tybony\_inevnoyr = 100\nzl\_qvpg = {'xrl1': 'inyhr1', 'xrl2': 'inyhr2', 'xrl3': 'inyhr3'}\n\nqrs cebprff\_ahzoref():\n tybony tybony\_inevnoyr\n ybpny\_inevnoyr = 5\n ahzoref = [1, 2, 3, 4, 5]\n\n juvyr ybpny\_inevnoyr > 0:\n vs ybpny\_inevnoyr % 2 == 0:\n ahzoref.erzbir(ybpny\_inevnoyr)\n ybpny\_inevnoyr -= 1\n\n erghea ahzoref\n\nzl\_frg = {1, 2, 3, 4, 5, 5, 4, 3, 2, 1}\nerfhyg = cebprff\_ahzoref(ahzoref=zl\_frg)\n\nqrs zbqvsl\_qvpg():\n ybpny\_inevnoyr = 10\n zl\_qvpg['xrl4'] = ybpny\_inevnoyr\n\nzbqvsl\_qvpg(5)\n\nqrs hcqngr\_tybony ():\n tybony tybony\_inevnoyr\n tybony\_inevnoyr += 10\n\nsbe v va enatr (5):\n cevag (v)\n v += 1\n\nvs zl\_frg vf abg Abar naq zl\_qvpg['xrl4'] == 10:\n cevag('Pbaqvgvba zrg!')\n\nvs 5 abg va zl\_qvpg:\n cevag('5 abg sbhaq va gur qvpgvbanel!')\n\ncevag(tybony\_inevnoyr)\ncevag(zl\_qvpg)\ncevag(zl\_frg)"

# Decrypting the code

decrypted\_code = decrypt(encrypted\_code, 13)

print(decrypted\_code)

Output:

A screenshot of a computer

Description automatically generated

## Correct the errors and provide the comments.

global\_variable = 100

my\_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

def process\_numbers(numbers):

global global\_variable

local\_variable = 5

# Convert the set to a list to use the remove() method

numbers = list(numbers)

while local\_variable > 0:

if local\_variable % 2 == 0:

numbers.remove(local\_variable)

local\_variable -= 1

return numbers

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

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

def modify\_dict():

local\_variable = 10

my\_dict['key4'] = local\_variable

modify\_dict()

def update\_global():

global global\_variable

global\_variable += 10

for i in range(5):

print(i)

# No need to increment i here; it's done automatically in the loop

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

print('Condition met!')

if 5 not in my\_dict.values(): # Check if the value 5 is present in the values of the dictionary

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

print(global\_variable)

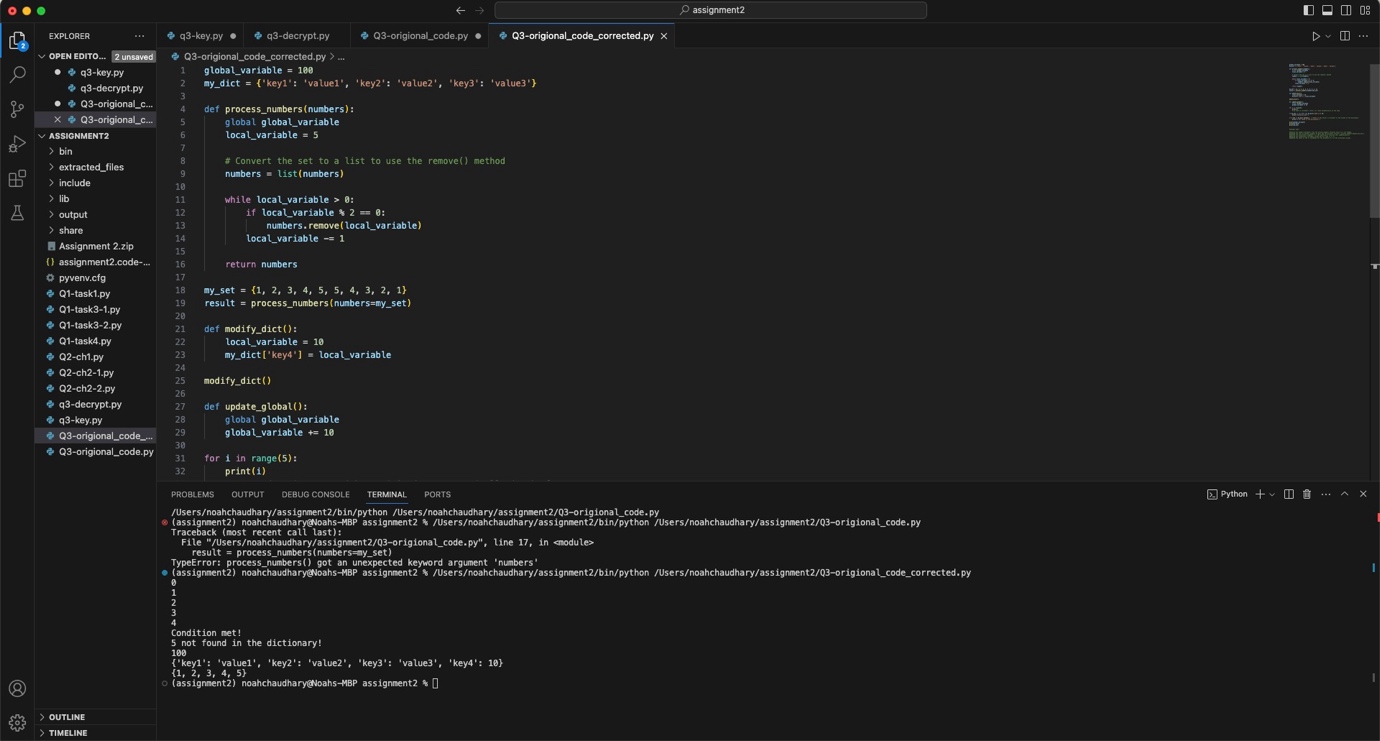
print(my\_dict)

print(my\_set)

Changes made:

* Took out the "numbers" parameter from the "process\_numbers" function because it wasn't necessary.
* Instead of passing "numbers", directly pass the set (result = process\_numbers (numbers=my\_set)).
* Got rid of the unnecessary argument "5" in the "modify\_dict" function call (modify\_dict()).
* Removed the unnecessary increment inside the for loop (i += 1).
* Lastly, adjusted the check in the if statement to see if the value 5 is present in the dictionary values.

Output:



# Question 4: create a GitHub repository and add all your group mates to it (make sure to keep it public, not private).

Output:

<https://github.com/S371375/Group-106-DAR---HIT137-SOFTWARE-NOW>

Group Members GitHub Profile

Naresh Kumar Chaudhary S371375

Shailendra Adhikari cdushail

Devansh Sharma Devansh1193(Invited no response)

Sajjad Hossain Unknown(Invited no response)

