**UNIT CODE: HIT137**
**Software Now**

# Assignment-2

## Submitted By

## Group: CAS-119


| Student Name | Student ID | Email |
|---|---|---|
| Synthia Islam | # S375728 | S375728@students.cdu.edu.au |
| Hussein Salami | #S371192 | S371192@students.cdu.edu.au |
| Sayeed Anwar | #S384116 | S384116@students.cdu.edu.au |
| A K M Shafiur Rahman | #S372618 | S372618@students.cdu.edu.au |

## Table of Contents

## Question 1: - [Sayeed Anwar, A K M Shafiur Rahman, Synthia Islam]

This question consists of multiple CSV files (In the Zipped Folder) with 'large texts' in one of the columns in each file. Your job is to 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'.

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).

Task 3: Programming and Research

3.1:

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.

3.2:

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.

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.)

### ANSWER:

**(Code is Included in "Question_1_All_task.py" file, attached in the zip file)**

#### Task 1 - Extracting Text from Multiple CSV Files:

We have extracted the large texts from the column containing the 'text' of the CSV files from the current directory where all given CSV files are stored. In this step, we are dealing with raw data that is distributed across multiple CSV files. Our goal is to extract the textual information, clean it, and prepare it for analysis. After extracting, cleaning, and preparing the texts, we have stored them in the "combined_texts.txt" file in the "output" folder.

```python
@staticmethod
def extract_and_clean_texts(dfs):
    """Extract and clean texts from a list of DataFrames and return a generator of cleaned texts."""
    for df, filename in dfs:  # Adjusted to unpack tuple
        # Identify columns containing 'TEXT' regardless of case
        text_column = next((col for col in df.columns if 'TEXT' in col.upper()), None)

        if text_column is not None:
            print(f"Extracting texts from '{text_column}' column in {filename}...")
            for text in df[text_column].dropna():
                yield TextProcessor.clean_text(text)
        else:
            print(f"No column name containing 'TEXT' found in {file_path}")

@staticmethod
def clean_text(text):
    """Clean text by removing non-English characters and extra spaces."""
    cleaned_text = re.sub(r'[^a-zA-Z\s]', '', text)
    cleaned_text = re.sub(r'\s+', ' ', cleaned_text)
    return cleaned_text.strip()

def process_texts(self, dfs):
    """Process and clean texts from CSV files and save the combined cleaned text."""
    if not dfs:
        print("No data available for processing")
    else:
        self.create_output_directory(self.output_directory)

        with open(self.output_path, 'w', encoding='utf-8') as outfile:
            for cleaned_text in self.extract_and_clean_texts(dfs):
                outfile.write(cleaned_text + '\n')
            print("'combined_texts.txt' file is created")
```

```python
output_directory = './output'
output_file = 'combined_texts.txt'

processor = TextProcessor(output_directory, output_file)
dfs = processor.extract_csv_to_dataframe()
processor.process_texts(dfs)
```

### Task 2 – Installing Libraries:

We have installed all the necessary libraries:

- spacy
- scispacy
- en_core_sci_sm
- en_ner_bc5cdr_md
- transformers

```python
# Task 2: Install the necessary libraries

print("Installing...")
# !pip install spacy
# !pip install scispacy
# !pip install https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.5.0/en_core_sci_sm-0.5.0.tar.gz
# !pip install https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.5.0/en_ner_bc5cdr_md-0.5.0.tar.gz
# !pip install transformers
print("Done.")
```

We have commented out the install command so that there is no interruption in running the code.

## Task 3 – Programming & Research:

### 3.1 - Count the Top 30 most common words:

We have used the WordAnalyzer class to perform text analysis by counting the most frequent words in a text file named “combined_texts.txt” obtained from the first step and saving the results to a CSV file named “top_30_words.csv” in the “output” folder.

```python
class WordAnalyzer:
    '''Word Analyzer Class'''
    def __init__(self, file_path, chunk_size=1024*1024, word_number=30):
        self.file_path = file_path
        self.chunk_size = chunk_size
        self.word_number = word_number

    def count_words(self):
        """Count the occurrences of words in the file."""
        word_counter = Counter()

        with open(self.file_path, 'r', encoding='utf-8') as infile:
            while True:
                chunk = infile.read(self.chunk_size)
                if not chunk:
                    print("No chunk is remaining!")
                    break

                # Process chunk
                words = chunk.split()
                word_counter.update(words)
                print("Word chunk is processing...")

        return word_counter.most_common(self.word_number)

    @staticmethod
    def save_to_csv(top_words, output_csv_path):
        """Save the top words and their counts to a CSV file."""
        with open(output_csv_path, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(['Word', 'Count'])
            writer.writerows(top_words)
        print("Top words and their counts are saved to 'top_30_words.csv' file!")
```

```python
    def process(self, output_csv_path):
        """Run the word count and save the results to a CSV file."""
        top_words = self.count_words()
        self.save_to_csv(top_words, output_csv_path)


# Word Analyzer Result
file_path = './output/combined_texts.txt'
output_csv_path = './output/top_30_words.csv'

analyzer = WordAnalyzer(file_path)
analyzer.process(output_csv_path)
pd.read_csv('./output/top_30_words.csv').head(5)
```

### 3.2 – Use AutoTokenizer from Transformers and Count 30 most common tokens:

We have used the “TokenAnalyzer” class to tokenize text data from a specified file “combined_texts.txt” obtained from the first step, count the occurrences of each token, and then record the top most frequent tokens in a CSV file named “top_30_tokens.csv” in the “output” folder. This class uses powerful NLP tools like BERT for tokenization, defaulting to "bert-base-uncased". This tokenizer is loaded from the transformer’s library. Here, “chunk_size” determines the size of the text chunks that the file is divided into for processing, with the default value set to 512 characters. The “count_tokens” method opens the specified file and reads it in chunks of defined size. The “token_number” is the number of top tokens to return and save. This “TokenAnalyzer” class provides an efficient way to handle large text files through chunk-based processing.

```python
class TokenAnalyzer:
    '''Token Analyzer Class'''
    def __init__(self, file_path, tokenizer_name="bert-base-uncased", chunk_size=512, token_number=30):
        self.file_path = file_path
        self.tokenizer = AutoTokenizer.from_pretrained(tokenizer_name, clean_up_tokenization_spaces=True)
        self.chunk_size = chunk_size
        self.token_number = token_number

    def count_tokens(self):
        """Count the occurrences of tokens in the file."""
        token_counts = Counter()

        with open(self.file_path, 'r', encoding='utf-8') as infile:
            while True:
                chunk = infile.read(self.chunk_size)
                if not chunk:
                    print("No chunk is remaining!")
                    break

                # Process chunk
                tokens = self.tokenizer.tokenize(chunk)
                token_counts.update(tokens)
                print("Token chunks are being updated!")

        return token_counts.most_common(self.token_number)

    @staticmethod
    def save_to_csv(top_tokens, output_csv_path):
        """Save the top tokens and their counts to a CSV file."""
        with open(output_csv_path, 'w', newline='', encoding='utf-8') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(['Token', 'Count'])
            writer.writerows(top_tokens)
        print("Top tokens and their counts are saved to 'top_30_tokens.csv' file!")
```

```
    def process(self, output_csv_path):
        """Run the token count and save the results to a CSV file."""
        top_tokens = self.count_tokens()
        self.save_to_csv(top_tokens, output_csv_path)


# Token Analyzer Result
file_path = './output/combined_texts.txt'
output_csv_path = './output/top_30_tokens.csv'

analyzer = TokenAnalyzer(file_path)
analyzer.process(output_csv_path)
pd.read_csv('./output/top_30_tokens.csv').head(5)
```

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

We have used "EntityAnalyzer" class that performs Named-Entity Recognition (NER) on textual data using both the spaCy and BioBERT models. This class is designed to handle large text files efficiently, process them to extract specific entities (diseases and drugs), and then compare and save the results.

- Load Models: It loads a spaCy model (en_ner_bc5cdr_md) for recognizing drug and disease entities and initializes BioBERT models for disease and drug NER, specifically trained on relevant datasets.
- Set up NER Pipelines: Then, it sets up NER pipelines using the loaded models and tokenizers, configured to run on available hardware (CPU or GPU).
- SpaCy Processing (_process_text_with_spacy): It extracts diseases and drugs from text chunks using spaCy, updating frequency counts for each entity.
- BioBERT Processing (_process_text_with_biobert): It processes text chunks with BioBERT models to identify and count disease and drug entities.
- Processing Text File (process_text_file): It reads the text file in designated chunks and applies both spaCy and BioBERT processing methods to each chunk.
- Get Results (get_results): Compiles and formats the counts of detected entities and their most common occurrences from both models.
- Save Results (save_results_to_files): The extracted entity data is saved into structured text files within specified directories for each model.
- Compare and Save Results (compare_and_save_results): It compares the detection results between spaCy and BioBERT, summarizing differences in entity detection frequencies in a comparative output file named "entity_analysis_results" in the "output" folder.

## Question 2: - [Synthia Islam]

Here's an adventurous story intertwined with Python programming questions that involve nested for loops, conditional statements, string manipulations, and more.

**The Quest for the Hidden Treasure:**

Deep within the mystical lands of Pythoria lay the fabled Temple of Codes, rumored to house a treasure of knowledge guarded by enigmatic puzzles. The path is challenging, and only those who can do the coding will unravel the final word, leading to the treasure.

**Chapter 1: The Gatekeeper**

```
import time

current_time = int(time.time())

generated_number = (current_time % 100) + 50

if generated_number % 2 == 0:
    generated_number += 10

print(generated_number)
```

The above algorithm generates a number (n). You should use this number to change the pixels (r,g,b) in the provided image (Chapter1.png) by adding the original pixel values (r,g,b) with the generated number (Example: (r+n, g+n, b+n)).

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.

**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 theeven numbers in the 'number substring' and upper-case letter in the 'letter string'
into ASCII Code Decimal Values.
Example Scenario:String = '56aAww1984sktr235270aYmn145ss785fsq31D0'
                 Separate them - 56198235270145785310 (number string) and aAwwsktraYmnssfsqD (letter string).
                Convert the even numbers in the number string to ASCII Code Decimal Values
                6, 8, 2, 2, 0, 4, 8, 0 (Even Numbers)
                54, 56, 50, 50, 48, 52, 56, 48 (ASCII CODE)
                Convert the upper-case letter in letter string to ASCII Code Decimal Values.
                A, Y, D (Upper-case Letters)
                65, 89, 68 (ASCII CODE Decimal Values)
```

**Chapter 3: Decrypt Cryptogram**

```
You are required to create a program that showcases the required output for the following question:
Many newspapers publish a cryptogram each day, for instance:

    VZ FRYSVFU VZCNGVRAG NAQ N YVGGYR VAFRPHER V ZNXR ZVFGNXRF 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

    The deciphered cryptogram is usually a quote from a famous author or celebrity.
    To get the original quote, you should replace each character in the ciphered quote using a shift keyvalue (s) condition.
    Example 1: If ciphered quote is AB, and 's' is 1, then original quote is ZA
    Example 2: If ciphered quote is AB, and 's' is 2, then original quote is YZ

    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.
```

## ANSWER:

**(Code is Included in "Ques_2.py" file, attached in the zip file along with the output file "chapter1out.png")**

### Chapter 1: The Gatekeeper

This chapter involves generating a number based on the current time and modifying an image's pixel values using this number.

- At first, we opened the provided image file "chapter1.jpg" and modified the pixels using the generated number. We have added error handling to ensure the image file can be opened and processed without issues
- Then, we saved the new image as "chapter1out.png".
- We also calculated the sum of the red pixel values.

```python
def modify_image(file_path):
    '''Chapter 1: Convert Image and Calculate Red Pixel Values'''
    try:
        # Generate the number
        current_time = int(time.time())
        generated_number = (current_time % 100) + 50
        if generated_number % 2 == 0:
            generated_number += 10

        # Open an image and modify it
        img = Image.open(file_path)
        img = img.convert('RGB')  # Ensure it's in RGB format
        pixels = img.load()

        # Modify the pixels
        for i in range(img.width):
            for j in range(img.height):
                r, g, b = pixels[i, j]
                pixels[i, j] = ((r + generated_number) % 256,
                                (g + generated_number) % 256,
                                (b + generated_number) % 256)

        # Save the new image
        img.save('./chapter1out.png')

        # Calculate the sum of the red pixel values
        red_sum = sum(pixels[i, j][0] for i in range(img.width) for j in range(img.height))
        return red_sum
    except IOError:
        print("Error: The file could not be opened or found.")
    except Exception as exc:
        print(f"An error occurred: {exc}")

    return 0
```

## Chapter 2: The Chamber of Strings

- For this challenge, we first split the string into numbers and letters.
- Then we converted even numbers to their ASCII values, and the same for upper-case letters.
- Finally, we returned the combined result of the ASCII values of even numbers and upper-case letters.

```python
# Chapter 2: The Chamber of Strings
def process_string(long_str):
    '''Chapter 2: Separate Even number and strings and convert them to ASCII Values'''
    if len(long_str) < 16:
        raise ValueError("String must be at least 16 characters long")

    # Separate numbers and letters
    number_string = ''.join(filter(str.isdigit, long_str))
    print("Number String: " + number_string)

    letter_string = ''.join(filter(str.isalpha, long_str))
    print("Letter String: " + letter_string + "\n")

    # Separate Even Numbers and Convert to their ASCII values
    even_num_string = ""
    ascii_numbers = []
    for num in number_string:
        if int(num) % 2 == 0:
            even_num_string += num + " "
            ascii_numbers += [str(ord(num))]

    print("Even Numbers: " + even_num_string)
    print("ASCII Values of Even Numbers: " + str(ascii_numbers) + "\n")

    # Separate Upper-case letters and Convert to their ASCII values
    upper_string = ""
    ascii_upper = []
    for letter in letter_string:
        if letter.isupper():
            upper_string += letter + " "
            ascii_upper += [str(ord(letter))]
```

```python
    print("Upper Case Letters: " + upper_string)
    print("ASCII Values of Upper Case Letters: " + str(ascii_upper) + "\n")
    ascii_uppercase = [str(ord(char)) for char in letter_string if char.isupper()]

    return ascii_numbers + ascii_uppercase
```

### Chapter 3: Decrypt a Cryptogram

- For the final part, we implemented a decryption algorithm to decrypt the given cryptogram.
    - We have normalized shift value to handle larger numbers.
    - We have tried different shifts to manually check which is the appropriate shift key to find out the meaningful original quote.
    - We have decrypted only alphabetical letters. Non-alphabetical letters are not changed.

```python
# Chapter 3: Decrypting a Cryptogram
def decipher_cryptogram(text, shift):
    '''Decrypt the cryptogram and find the original quote and shift value'''
    if not text.strip():  # Check if text is not empty or just whitespace
        raise ValueError("Text must not be empty")

    shift = shift % 26  # Normalize shift to handle larger numbers

    decrypted_text = []
    for char in text:
        if char.isalpha():  # Only decrypt alphabetical characters
            shifted = ord(char) - shift
            if char.isupper():
                if shifted < ord('A'):
                    shifted += 26
            elif shifted < ord('a'):
                shifted += 26
            decrypted_text.append(chr(shifted))
        else:
            decrypted_text.append(char)  # Non-alphabetic characters are not changed
    return ''.join(decrypted_text)
```

Main Code:

```python
def main():
    '''Main function'''
    # Chapter 1: Example for Chapter1.jpg
    print("Chapter 1:")

    red_pixel_sum = modify_image('./chapter1.jpg')
    print(red_pixel_sum)

    # Chapter 2: Example for Separating Upper-case letters and Converting to their ASCII values
    print("\nChapter 2:")
    try:
        input_string = "56aAww1984sktr235270aYmn145ss785fsq31D0"
        result = process_string(input_string)
        print("The Combined Result: " + str(result))
    except Exception as e:
        print(e)

    # Chapter 3: Decrypt the given cryptogram and Find out the original quote with the shift key
    print("\nChapter 3:")
    try:
        cipher_text = "VZ FRYSVFU VZCNGVRAG NAQ N YVGGYR VAFRPHER V ZNXR ZVFGNXRF V NZ BHG BS PBAGEBY NAQ NG GVZRF UNEQ GB UNAQYR OHG VS

        # Trying different shifts
        for shift in range(26):
            decrypted_message = decipher_cryptogram(cipher_text, shift)
            print(f"Shift {shift}: {decrypted_message}")
        print("")
        print("From the result, we have found that the Shift Key is 13\n")
        decrypted_text = decipher_cryptogram(cipher_text, 13)
        print("Original Quote is: " + decrypted_text)
    except Exception as exp:
        print(exp)
```

```python
# Driver code
if __name__ == '__main__':
    # Example of The Quest for the Hidden Treasure
    main()
```

## Question 3: - [Hussein Salami, Synthia Islam]

Fixing the error-prone codes. Below is the code (left) that is encrypted using a number. Once you decrypt the below code, it reveals the original code with many errors. Please fix them and explain them using comments (#).

```python
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 hcqngr_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)
```

```python
def encrypt(text, key):
    encrypted_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
            encrypted_text += chr(shifted)
        else:
            encrypted_text += char
    return encrypted_text


key = ???????????????
encrypted_code = encrypt(original_code, key)
print(encrypted_code)
```

To decrypt the above code, first, you need to understand how it is encrypted (above right image).

1. Fixing the next code will reveal the key.
2. Write the decryption function to decrypt the ‘encrypted code’ to the original code.
3. Correct the errors and provide the comments.
4. Should show everything in your program file.

```
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
```

## Answer:

- At first, we have found the shift key from the given code in order to decrypt the code.

```
def find_key():
    '''Find Key Function'''
    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("\nKey: " + str(total))
    return total
```

- Then, we decrypted the code using the shift key found in the previous step. We decrypted only alphabetical letters. Non-alphabetical letters were unchanged.

```python
def decrypt(encrypted_text, key):
    '''decrypt method'''
    decrypted_text = ""
    for char in encrypted_text:
        if char.isalpha():  # Check if the character is alphabetical
            shifted = ord(char) - key
            if char.islower():  # Wrap around for lowercase letters
                if shifted < ord('a'):
                    shifted += 26
            elif char.isupper():  # Wrap around for uppercase letters
                if shifted < ord('A'):
                    shifted += 26
            decrypted_text += chr(shifted)
        else:
            decrypted_text += char  # Non-alphabetic characters are unchanged
    return decrypted_text
```

- There were several errors in the original code which we got from the decryption result.
- Then we corrected those errors and wrote comments on those places:
  - Added method docstring.
  - In “process_numbers” method:
    - Removed global definition of “global_variable” in as there is no use of it.
    - We added ‘numbers’ as a parameter of the main method.
    - Removed initialization of 'numbers' as we have used it as a parameter.
  - Sets automatically remove duplicates, so the initial set was defined with duplicates unnecessarily. Hence, we removed the redundant duplicate data during its initialization.
  - Removed the erroneous parameter from “modify_dict()” method.
  - update_global() method was not called from anywhere. So, we correctly called the function to update global variable.
  - In the for loop of range(5), incrementing 'i' inside the loop has no effect due to the nature of Python loops, so we have removed it.

```python
# Correct the Errors and Provide the Comments in the Original Code
global_variable = 100
my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}


def process_numbers(numbers):  # Added 'numbers' as a parameter
    '''process_numbers function'''  # Added Function Docstring
    # Removed global definition of global_variable as there is no use of it
    # global global_variable
    local_variable = 5
    # Removed initialization of 'numbers' as we have used it as a parameter
    # numbers = [1, 2, 3, 4, 5]

    while local_variable > 0:
        if local_variable % 2 == 0:
            numbers.remove(local_variable)
        local_variable -= 1

    return numbers


# Sets automatically remove duplicates, initial set defined with duplicates unnecessarily
# my_set = {1, 2, 3, 4, 5, 5, 4, 3, 2, 1}
my_set = {1, 2, 3, 4, 5}
result = process_numbers(numbers=my_set)


def modify_dict():
    '''Method of Modifying Dictionary'''  # Added Method Docstring
    local_variable = 10
    my_dict['key4'] = local_variable

💡
modify_dict() # Removed the erroneous parameter
```

```python
def update_global():
    '''Method of Updating Global variable'''  # Added Method Docstring
    global global_variable
    global_variable += 10


update_global()  # Correctly call the function to update global variable


for i in range(5):
    print(i)
    # Incrementing 'i' inside the loop has no effect due to the nature of Python loops, so removing it
    # 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(my_dict)
print(my_set)
```

## Question 4: - [Synthia Islam]

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.

## Answer:

GitHub public repository is created for the team, and members are properly added:

**Link:** https://github.com/cas119/HIT137-software-now-cas119