Software security assignment 2 - Group 15

Anisha Das (adas97)
Jayagauri Adinath Sunke (jsunke)
Mohammad Danish Khan (mkhan118)

To implement code for Ques 1 and 2 (Java)

- Ensure Java is configured
- Run the corresponding files for Ques 1: swsec_q1
- Run the corresponding files for Ques 2: swsec_q2

To implement code for Ques 3 and 4 (Python)

- Ensure Python3 is configured
- Run the corresponding files for Ques 3: vignere_cipher.py
- Run the corresponding files for Ques 4: vigenereKeyFinder.py

Ques 1

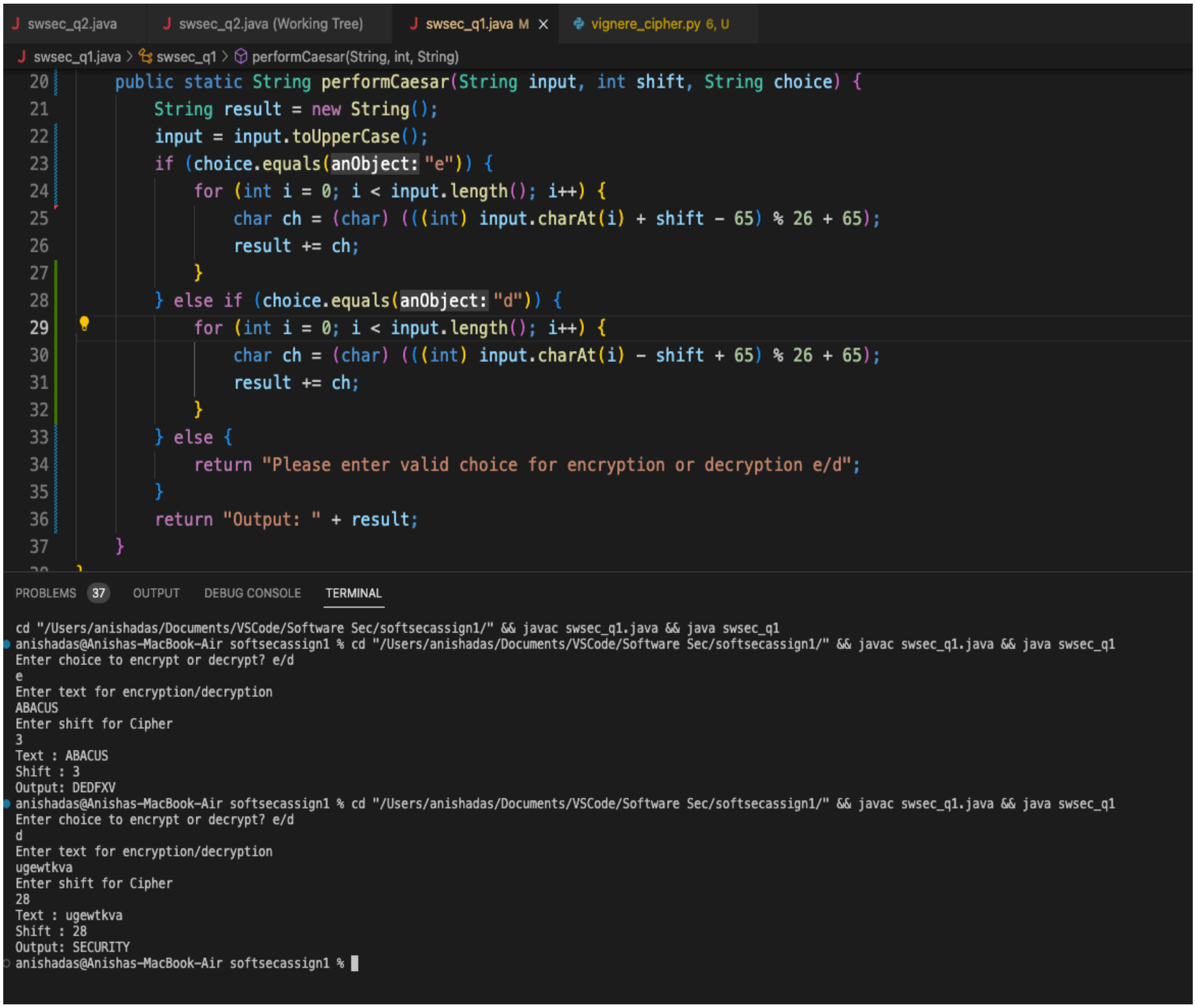
Explanation:
The program has been designed to work on strings with both uppercase and lowercase inputs to give the resulting ciphertext in only uppercase characters. The Caesar cipher is a monoalphabetic shift-based cipher, which 'shifts' each character in the plaintext with a specified key value to form the resulting ciphertext.
We have achieved the following logic in our code by implementing the 'performCaesar' function that takes the user input, the shift key and the choice to encrypt (e) or decrypt (d) as its parameters. We first convert the strings to all uppercase characters. To encrypt the input plaintext to ciphertext, we replace the character with the result of a function that calculates the resulting ciphertext character after the specified shift. We take the ASCII values of 'A' respectively and the shift requested, take mod with 26 and add it back to the ASCII values of 'A'.
Similarly, to decrypt the input ciphertext with the specific shift, we revert the mechanism and move the shift backwards on the characters to obtain the plaintext.
Note: The program written doesn't account for space, i.e. it considers the space character as an independent character and shifts it in the resultant cipher

Input:

1. e, abacus, shift: 5
2. d, ugewtkva, shift: 28

Output:

1. FGFHZX (encryption)
2. SECURITY (decryption)



```java
    public static String performCaesar(String input, int shift, String choice) {
        String result = new String();
        input = input.toUpperCase();
        if (choice.equals(anObject: "e")) {
            for (int i = 0; i < input.length(); i++) {
                char ch = (char) (((int) input.charAt(i) + shift - 65) % 26 + 65);
                result += ch;
            }
        } else if (choice.equals(anObject: "d")) {
            for (int i = 0; i < input.length(); i++) {
                char ch = (char) (((int) input.charAt(i) - shift + 65) % 26 + 65);
                result += ch;
            }
        } else {
            return "Please enter valid choice for encryption or decryption e/d";
        }
        return "Output: " + result;
    }
```

```
cd "/Users/anishadas/Documents/VSCode/Software Sec/softsecassign1/" && javac swsec_q1.java && java swsec_q1
anishadas@Anishas-MacBook-Air softsecassign1 % cd "/Users/anishadas/Documents/VSCode/Software Sec/softsecassign1/" && javac swsec_q1.java && java swsec_q1
Enter choice to encrypt or decrypt? e/d
e
Enter text for encryption/decryption
ABACUS
Enter shift for Cipher
3
Text : ABACUS
Shift : 3
Output: DEDFXV
anishadas@Anishas-MacBook-Air softsecassign1 % cd "/Users/anishadas/Documents/VSCode/Software Sec/softsecassign1/" && javac swsec_q1.java && java swsec_q1
Enter choice to encrypt or decrypt? e/d
d
Enter text for encryption/decryption
ugewtkva
Enter shift for Cipher
28
Text : ugewtkva
Shift : 28
Output: SECURITY
anishadas@Anishas-MacBook-Air softsecassign1 % 
```

Ques 2

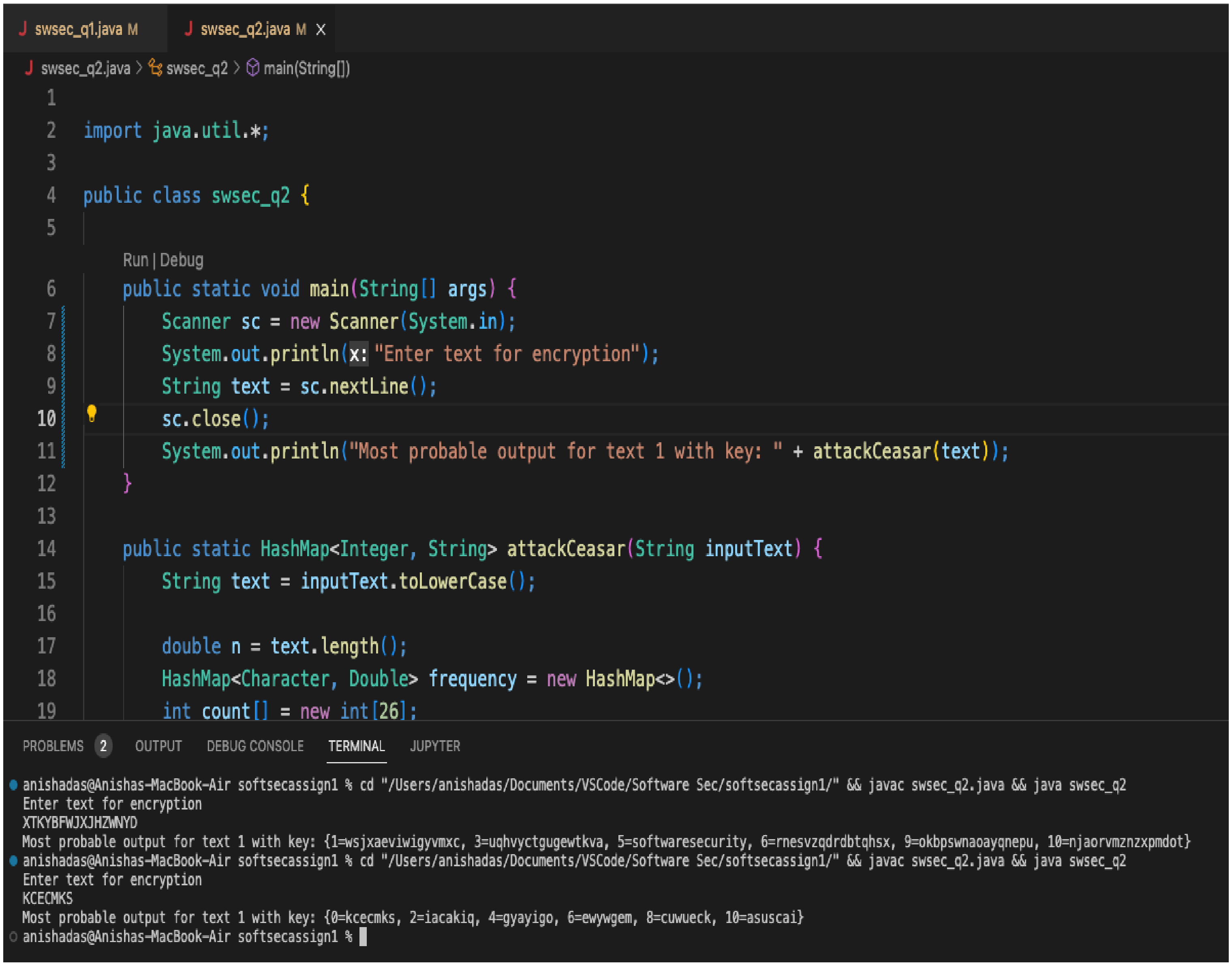
Explanation:
To attack the cipher using statistical analysis, we compute the frequency of each character in the input and store it in a map called 'frequency' by counting each occurrence of each letter and dividing it by the length of the input. We also store the one-gram frequency table in a map called 'oneGram'. We then create a correlation map of the individual correlations for each value of the key from 0 to 25 and store it in a key value pair in map 'value2idx'. Finally, we sort the correlation values in decreasing order and output the 6 most probable keys and plaintext values based on the highest correlation values obtained.

Input:
XTKYBFWJXJHZWNYD
KCECMKS

Output:
Most probable output for text 1 with key: {1=wsjxaeviwigyvmxc, 3=uqhvyctgugewtkva, 5=softwaresecurity, 6=rnesvzqdrdbtqhsx, 9=okbpswnaoayqnepu, 10=njaorvmznzxpmdot}

Most probable output for text 2 with key: {0=kcecmks, 2=iacakiq, 4=gyayigo, 6=ewywgem, 8=cuwueck, 10=asuscai}

Understanding the output:
We return a map of the pairs of the most probable solutions with the key and the plaintext pairs. Reviewing the output for the first input, the pair 5=softwaresecurity makes the most sense as a relevant English phrase. Hence the shift key is 5 and the plain text is softwaresecurity
Similarly, for the second input, 10=asuscai makes the most sense. Hence the shift key is 10 and the plain text is asuscai



```java

import java.util.*;

public class swsec_q2 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println(x: "Enter text for encryption");
        String text = sc.nextLine();
        sc.close();
        System.out.println("Most probable output for text 1 with key: " + attackCeasar(text));
    }

    public static HashMap<Integer, String> attackCeasar(String inputText) {
        String text = inputText.toLowerCase();

        double n = text.length();
        HashMap<Character, Double> frequency = new HashMap<>();
        int count[] = new int[26];
```

```
anishadas@Anishas-MacBook-Air softsecassign1 % cd "/Users/anishadas/Documents/VSCode/Software Sec/softsecassign1/" && javac swsec_q2.java && java swsec_q2
Enter text for encryption
XTKYBFWJXJHZWNYD
Most probable output for text 1 with key: {1=wsjxaeviwigyvmxc, 3=uqhvyctgugewtkva, 5=softwaresecurity, 6=rnesvzqdrdbtqhsx, 9=okbpswnaoayqnepu, 10=njaorvmznzxpmdot}
anishadas@Anishas-MacBook-Air softsecassign1 % cd "/Users/anishadas/Documents/VSCode/Software Sec/softsecassign1/" && javac swsec_q2.java && java swsec_q2
Enter text for encryption
KCECMKS
Most probable output for text 1 with key: {0=kcecmks, 2=iacakiq, 4=gyayigo, 6=ewywgem, 8=cuwueck, 10=asuscai}
anishadas@Anishas-MacBook-Air softsecassign1 %
```

Ques 3

Explanation:

- Encryption:

  Utilizing Vigenere, encryption is done with a letter-based key (and an alphabet). We take the first letter of the message and the first letter of the key, and add their values to cipher a text (letters have a value depending on their rank in the alphabet, starting with 0). The rank of the ciphered letter is determined by adding the results modulo 26 (26 is the number of letters in the alphabet).

  For example, plainText = SOFTWARESECURITY and the key = CSE.
  The first letters of the plaintext S (value = 18) and the key C (value = 2) are added together, we get the number 20 modulo 26 = 20 which corresponds to the alphabet U.

  Continue with the following letter of the key and the following letter of the plaintext. Return to the first letter of the key when you reach the end of it. In the end we get the cipherText = UGJVOETWWGUYTAXA. This example is for the standard Vigenere Cipher.
- Decryption:

  The first letter of the ciphertext and the first letter of the key are used to decrypt the message (letters have a value equal to their position in the alphabet starting from 0). We subtract the rank of the first letter of the ciphertext and the first letter of key, then we add 26(To counteract the negatives) and take modulo 26 of the result (26 is the total number of alphabetic letters), and the result will reveal the rank of the simple letter. Continue with the following letter of the key and the following letter of the ciphertext. Return to the first letter of the key when you reach the end of it.

  For example, cipherText = UGJVOETWWGUYTAXA and the key = CSE. The first letters of the cipherText U (value = 20) and the key C (value = 2) are subtracted together, we get the number (18 + 26) modulo 26 = 18 which corresponds to the alphabet S.

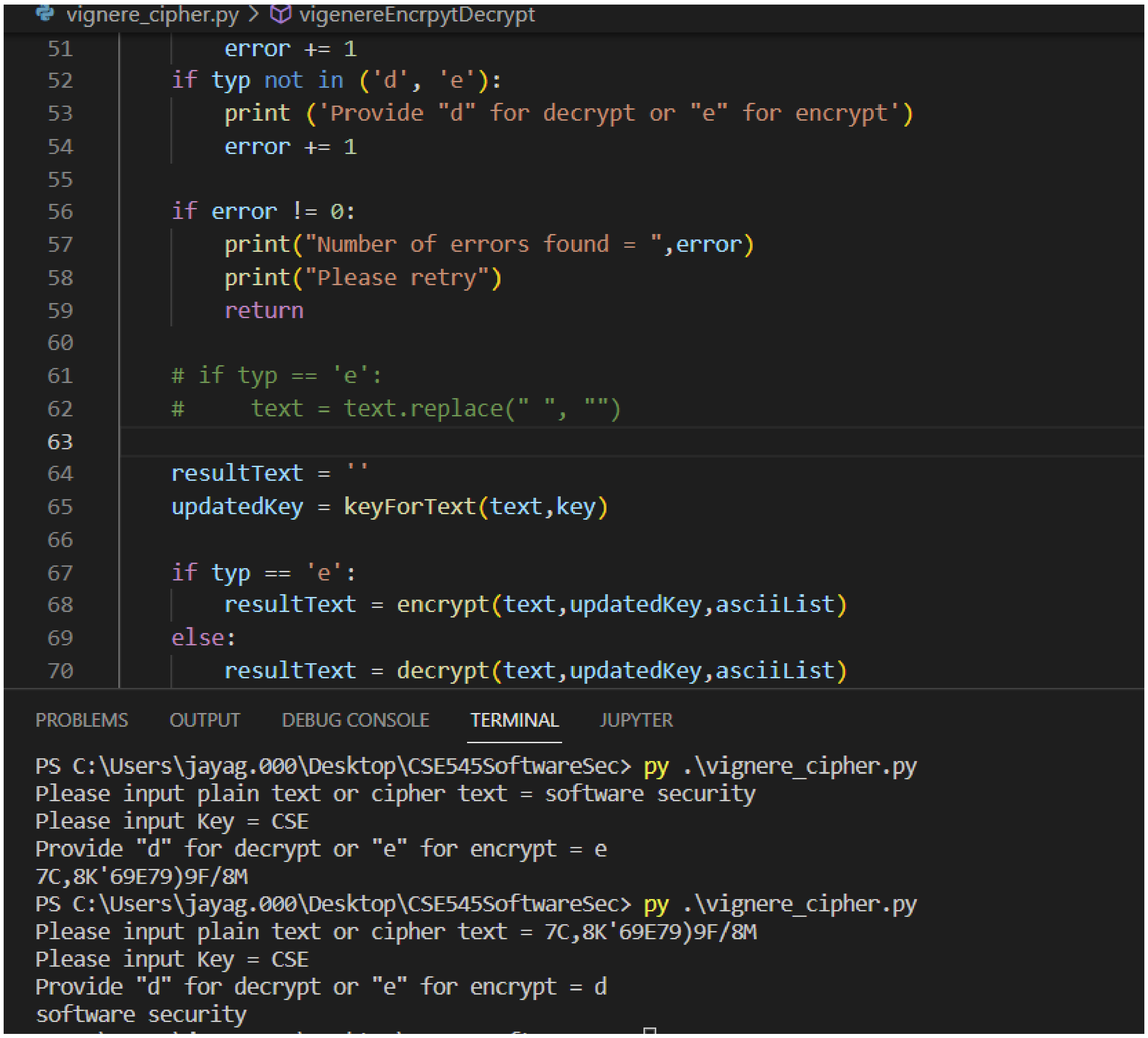
In the case of Q3 we have a variation where the list of characters is not restricted to just Alphabets, it can contain all ASCII characters. To implement this we first made a list of all ASCII characters. The only difference this made is that instead of the range of characters being from 0-25 the range increases to 0-len(asciiList)-1 and instead of modulo 26 we take modulo len(asciiList). Moreover there are checks given to see if the Key provided is between 1-3 characters and each character is upper-case alphabet. The code also contains a typ ‘e’ or ‘d’ to provide the user with both encryption or decryption abilities. The program takes into account of spaces and treats them as characters

Inputs :

- **Encryption :**
  PlainText = software security
  Key = CSE
- **Decryption :**
  CipherText = 7C,8K'69E79)9F/8M
  Key = CSE

Outputs :

- **Encryption :** 7C,8K'69E79)9F/8M
- **Decryption :** software security



vignere_cipher.py > vigenereEncrpytDecrypt

```
            error += 1
        if typ not in ('d', 'e'):
            print ('Provide "d" for decrypt or "e" for encrypt')
            error += 1

        if error != 0:
            print("Number of errors found = ",error)
            print("Please retry")
            return

        # if typ == 'e':
        #     text = text.replace(" ", "")

        resultText = ''
        updatedKey = keyForText(text,key)

        if typ == 'e':
            resultText = encrypt(text,updatedKey,asciiList)
        else:
            resultText = decrypt(text,updatedKey,asciiList)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
PS C:\Users\jayag.000\Desktop\CSE545SoftwareSec> py .\vignere_cipher.py
Please input plain text or cipher text = software security
Please input Key = CSE
Provide "d" for decrypt or "e" for encrypt = e
7C,8K'69E79)9F/8M
PS C:\Users\jayag.000\Desktop\CSE545SoftwareSec> py .\vignere_cipher.py
Please input plain text or cipher text = 7C,8K'69E79)9F/8M
Please input Key = CSE
Provide "d" for decrypt or "e" for encrypt = d
software security
```

Ques4

Explanation:
To find the Key given the Plaintext and Ciphertext, we need to check for key length 1, 2 and 3 since we know that the key length is between 1-3. One more restriction that we know about is that the ciphertext, plaintext and key are all from characters between “A” - “Z”.
We start the iteration taking key length as 1. We start checking from Key = “A”, use the same logic for decryption as above and check if “A” is able to decrypt the first character of the ciphertext or not. We do this by subtracting the rank of the first characters of ciphertext and key and check if its equal to the rank of the character of plaintext. If yes then that is our key. We then check the key for the other characters as well. If the decryption is successful, the key = actual key. If not we move on to the next iteration for key length 2 and keep doing the same steps till the Key is found.
Every key that is checked is added to the list of keys and written in the output text file.

Input:

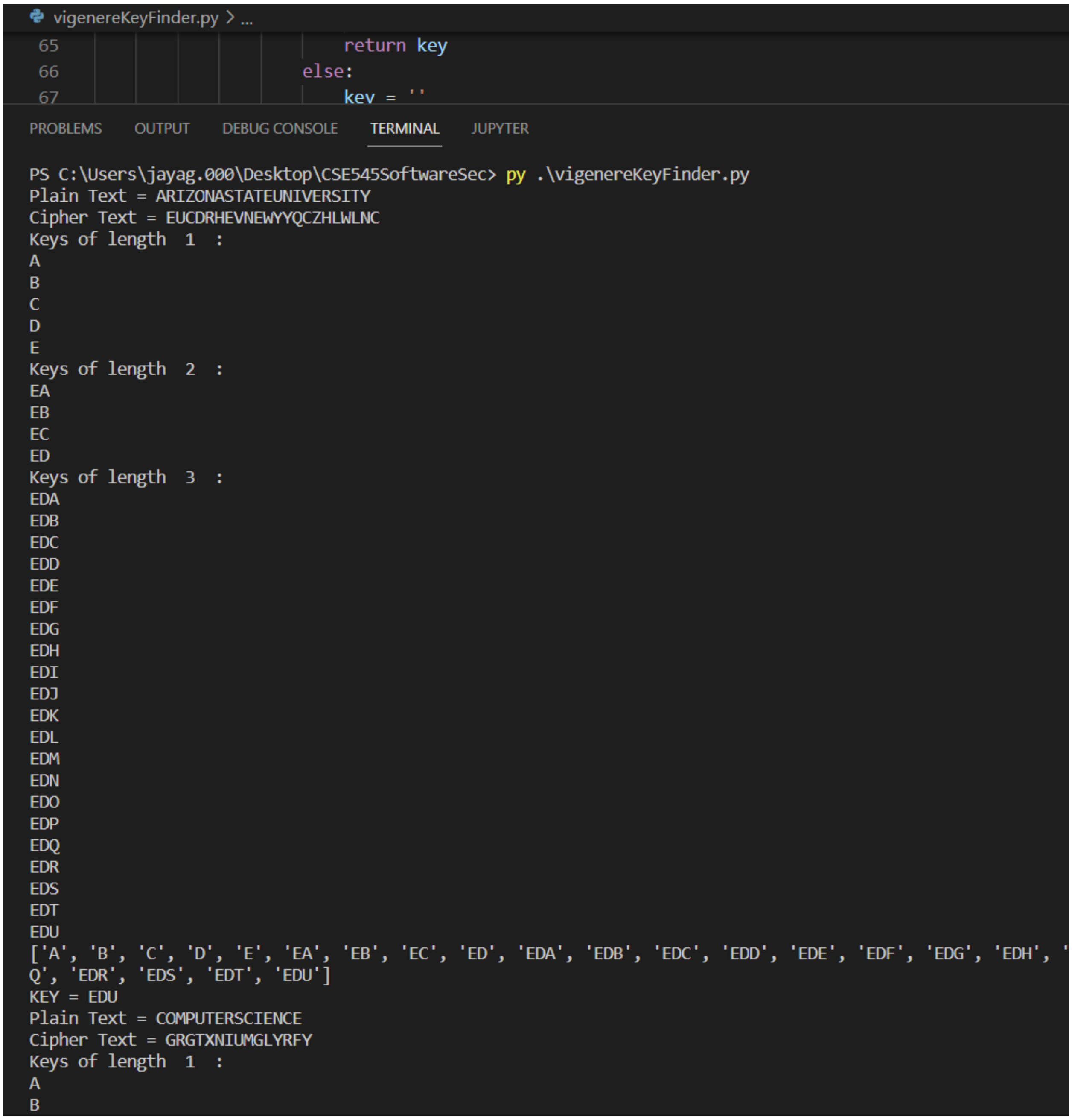
1) PlainText = ARIZONASTATEUNIVERSITY
   CipherText = EUCDRHEVNEWYYQCZHLWLNC

2) PlainText = COMPUTERSCIENCE
   CipherText = GRGTXNIUMGLYRFY

Ouput:

Output text file has been uploaded in the zip file (output.txt)

1) Key = EDU
2) Key = EDU

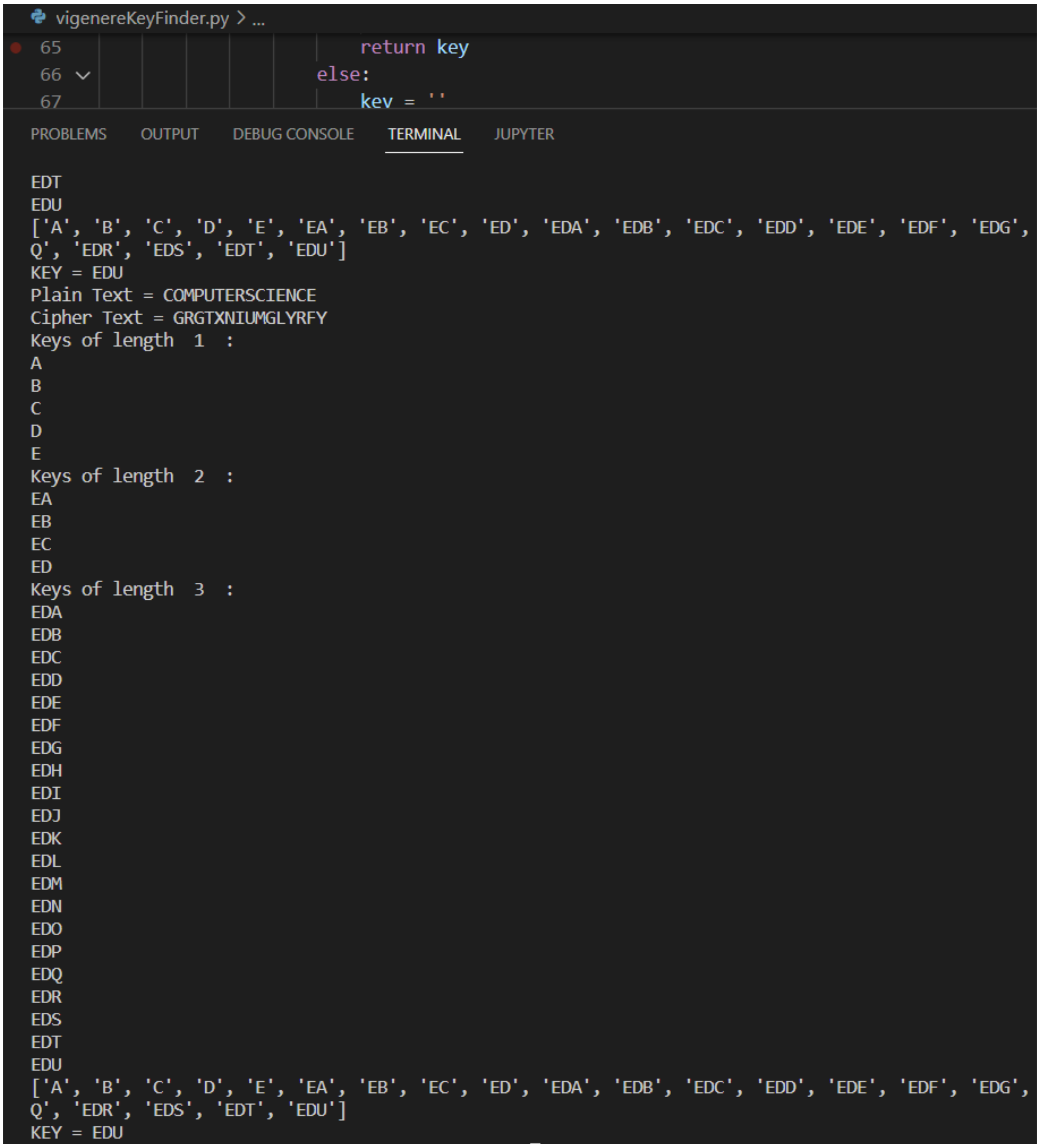


vigenereKeyFinder.py > ...

```
                    return key
                else:
                    key = ''
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
PS C:\Users\jayag.000\Desktop\CSE545SoftwareSec> py .\vigenereKeyFinder.py
Plain Text = ARIZONASTATEUNIVERSITY
Cipher Text = EUCDRHEVNEWYYQCZHLWLNC
Keys of length  1  :
A
B
C
D
E
Keys of length  2  :
EA
EB
EC
ED
Keys of length  3  :
EDA
EDB
EDC
EDD
EDE
EDF
EDG
EDH
EDI
EDJ
EDK
EDL
EDM
EDN
EDO
EDP
EDQ
EDR
EDS
EDT
EDU
['A', 'B', 'C', 'D', 'E', 'EA', 'EB', 'EC', 'ED', 'EDA', 'EDB', 'EDC', 'EDD', 'EDE', 'EDF', 'EDG', 'EDH', '
Q', 'EDR', 'EDS', 'EDT', 'EDU']
KEY = EDU
Plain Text = COMPUTERSCIENCE
Cipher Text = GRGTXNIUMGLYRFY
Keys of length  1  :
A
B
```



vigenereKeyFinder.py > ...

```
                    return key
                else:
                    key = ''
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

```
EDT
EDU
['A', 'B', 'C', 'D', 'E', 'EA', 'EB', 'EC', 'ED', 'EDA', 'EDB', 'EDC', 'EDD', 'EDE', 'EDF', 'EDG',
Q', 'EDR', 'EDS', 'EDT', 'EDU']
KEY = EDU
Plain Text = COMPUTERSCIENCE
Cipher Text = GRGTXNIUMGLYRFY
Keys of length  1  :
A
B
C
D
E
Keys of length  2  :
EA
EB
EC
ED
Keys of length  3  :
EDA
EDB
EDC
EDD
EDE
EDF
EDG
EDH
EDI
EDJ
EDK
EDL
EDM
EDN
EDO
EDP
EDQ
EDR
EDS
EDT
EDU
['A', 'B', 'C', 'D', 'E', 'EA', 'EB', 'EC', 'ED', 'EDA', 'EDB', 'EDC', 'EDD', 'EDE', 'EDF', 'EDG',
Q', 'EDR', 'EDS', 'EDT', 'EDU']
KEY = EDU
```