The report covers the following topics:

- **Code Structure and Explanation: Detailed descriptions of each class and their methods, explaining how the encryption, decryption, and hacking processes work.**
- **Encryption and Decryption Process: Step-by-step breakdown of how a message is encrypted by Alice, decrypted by Bob, and analyzed by Oscar.**
- **Frequency Analysis Attack: Explanation of how Oscar uses letter frequency analysis to attempt to crack the encrypted message, including the unique aspects of the English language that affect this process.**
- **Practical Output: Demonstration of the outputs from the encryption, decryption, and hacking processes to show the practical implementation and effectiveness of the system.**
- **Summary and Conclusion: A recap of the simulation, highlighting the key learnings and the importance of understanding encryption and decryption in the context of software optimization.**
- **Source Code: Finally, you will see the complete source code used for this project, providing a clear and practical example of the implementation.**

By the end of this report, you will have a comprehensive understanding of how a monoalphabetic cipher works, specifically tailored to the English language, and the potential vulnerabilities that can be exploited by attackers using frequency analysis.

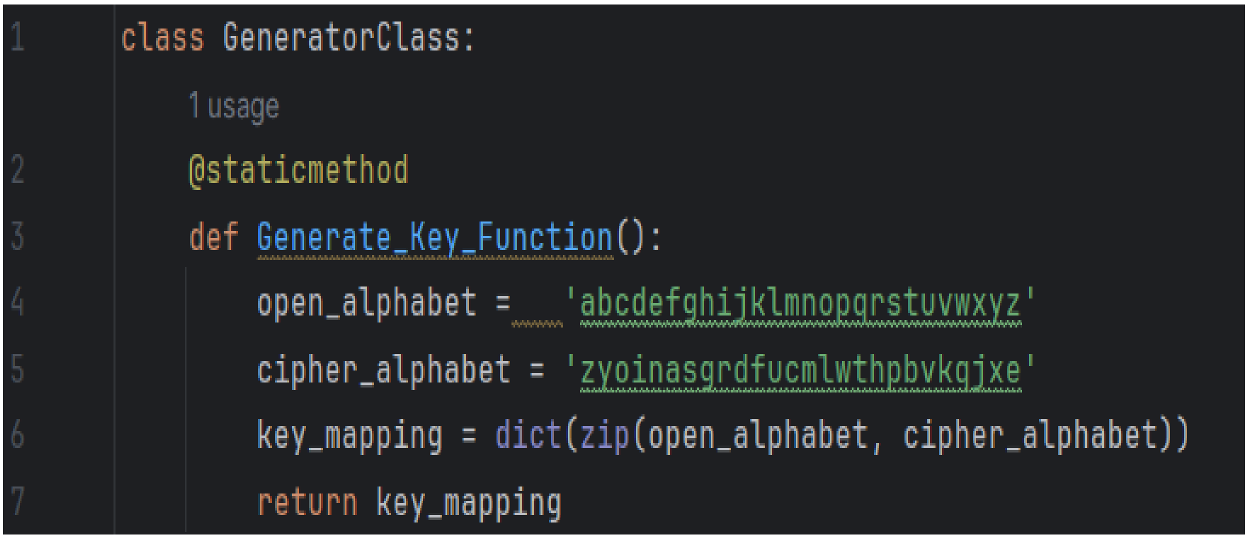
Code Structure

This code implements a simple encryption and decryption system. There are four main classes:

1. **GeneratorClass**: Generates a key mapping for encryption and decryption.
2. **ALiceEncrypterClass**: Encrypts a given message.
3. **BobDecrypterClass**: Decrypts an encrypted message.
4. **Oscar**: Receives the encrypted message and attempts to hack it using letter frequency analysis.
5. **main()**: Controls the main flow; generates the key, encrypts the message, decrypts the message, and simulates Oscar's attempt to hack the message.

GeneratorClass:

This class creates the **(key)** that will be used for the encryption process.



```
class GeneratorClass:
    1 usage
    @staticmethod
    def Generate_Key_Function():
        open_alphabet =  'abcdefghijklmnopqrstuvwxyz'
        cipher_alphabet = 'zyoinasgrdfucmlwthpbvkqjxe'
        key_mapping = dict(zip(open_alphabet, cipher_alphabet))
        return key_mapping
```

Here I created the encryption in **“GeneratorClass”**. The letters **“open_alphabet”** form the key that will replace the letters **“cipher_alphabet”.**

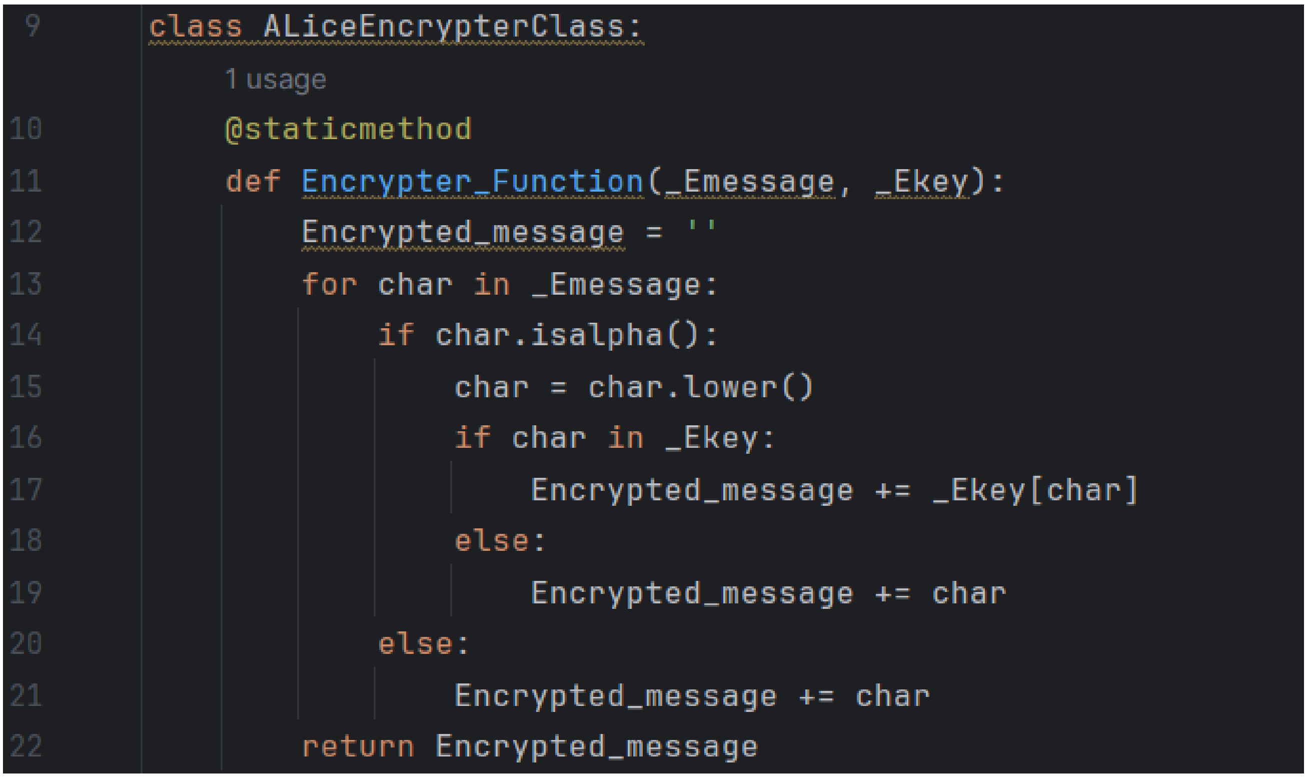
GeneratorClass:

Generate_Key_Function The method defines two alphabetic strings:

- **open_alphabet**: It consists of 26 letters of the English alphabet.
- **cipher_alphabet**: A complex sequence of letters representing the cryptic alphabet. And this will mix with **“open_alphabet”.**
- **key_mapping:** Th dictionary contains this mapping which maps clear letters to encrypted letters and is used for the encryption process.
- **The zip(open_alphabet, cipher_alphabet)** function maps these two arrays, and the **dict()** function returns these mappings as a dictionary. For example:
- **a -> z**
- **b -> y**
- **c -> o**
- **etc.**

ALiceEncrypterClass

This class encrypts a given message (**string**) using a key.



```python
class ALiceEncrypterClass:
    @staticmethod
    def Encrypter_Function(_Emessage, _Ekey):
        Encrypted_message = ''
        for char in _Emessage:
            if char.isalpha():
                char = char.lower()
                if char in _Ekey:
                    Encrypted_message += _Ekey[char]
                else:
                    Encrypted_message += char
            else:
                Encrypted_message += char
        return Encrypted_message
```

Here, I first start as an empty string, check each letter one by one with a **“for”** loop, and change the letters to lowercase each time. And I add it to the empty "string" by doing the conversion. If I find that there are no letters such as periods or commas, I detect it with **“char.isalpha():”** and add it to the **ciphertext** without changing it.

EncrypterFunction

- **Encrypter_Function** method takes two parameters:
- **_Emessage**: The message to be encrypted.
- **_Ekey**: Encryption key (dictionary).

- **Encrypted_message**: It starts as an empty string to create the encrypted message.
- **The for loop processes each character in _Emessage one by one:**

  • **char.isalpha():** Checks whether the character is a letter.

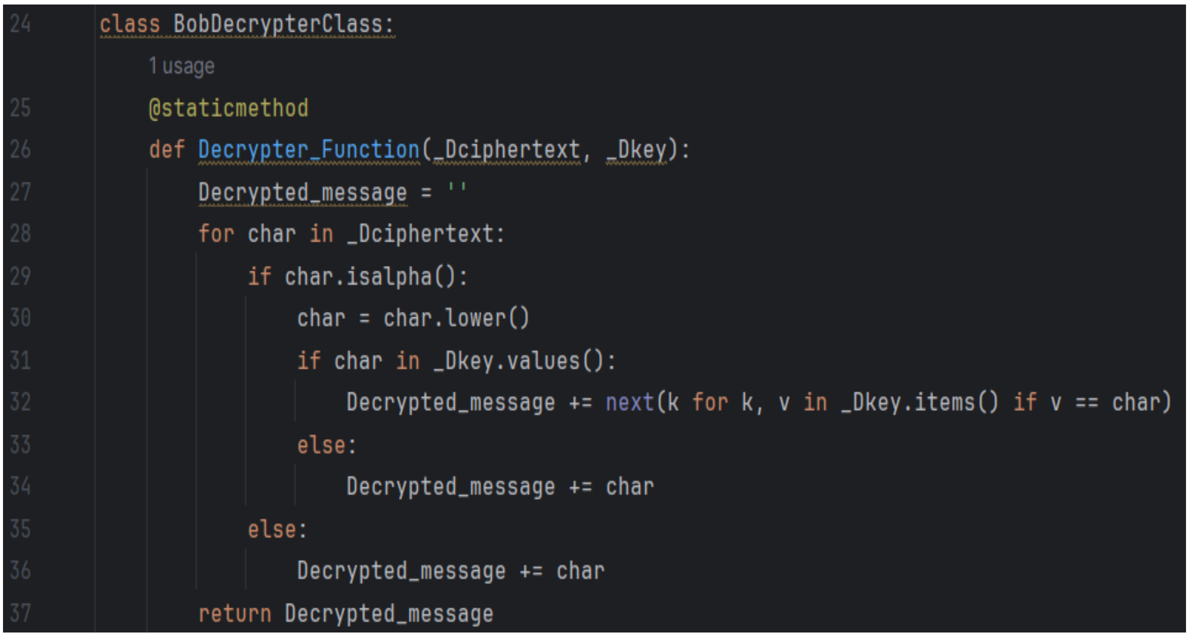
  • **char.lower():** Converts the character to lowercase (encryption is done in lowercase letters).

  • **If char** is present in the key **(_Ekey**), the corresponding encrypted letter is added to **Encrypted_message**. **Else**, the character is added as is (for example, punctuation marks or numbers).

  • **If char.isalpha()** returns false (that is, the character is not a letter), the character is added directly to the **Encrypted_message.**

  • After the encryption process is finished, **Encrypted_message** is returned.

BobDecrypterClassThis class decrypts a given encrypted message using a key.



```python
class BobDecrypterClass:
    @staticmethod
    def Decrypter_Function(_Dciphertext, _Dkey):
        Decrypted_message = ''
        for char in _Dciphertext:
            if char.isalpha():
                char = char.lower()
                if char in _Dkey.values():
                    Decrypted_message += next(k for k, v in _Dkey.items() if v == char)
                else:
                    Decrypted_message += char
            else:
                Decrypted_message += char
        return Decrypted_message
```

Here I am receiving the encrypted text from Alice. Then I also get my **"_Dkey"** decryption key. I create an empty string to print the decoded message and check each character one by one in the "for" loop. If the values match the **"_Dkey"** values, I find the original letter with this value with **"next()"** and add it to the empty **"string"**. Else, I add the character directly to the empty text.

- **Decrypter_Function** The method takes two parameters:
    - **_Dciphertext**: Encrypted message to be decrypted.
    - **_Dkey**: Decryption key (dictionary).
    - **Decrypted_message**: It starts as an empty string to create the decoded message.
    - **For loop** processes each character in **_Dciphertext** one by one**:**
    - **char.isalpha()**: Checks if the character is a letter.
    - **char.lower()**: Converts the character to lowercase (decryption is done in lowercase letters).

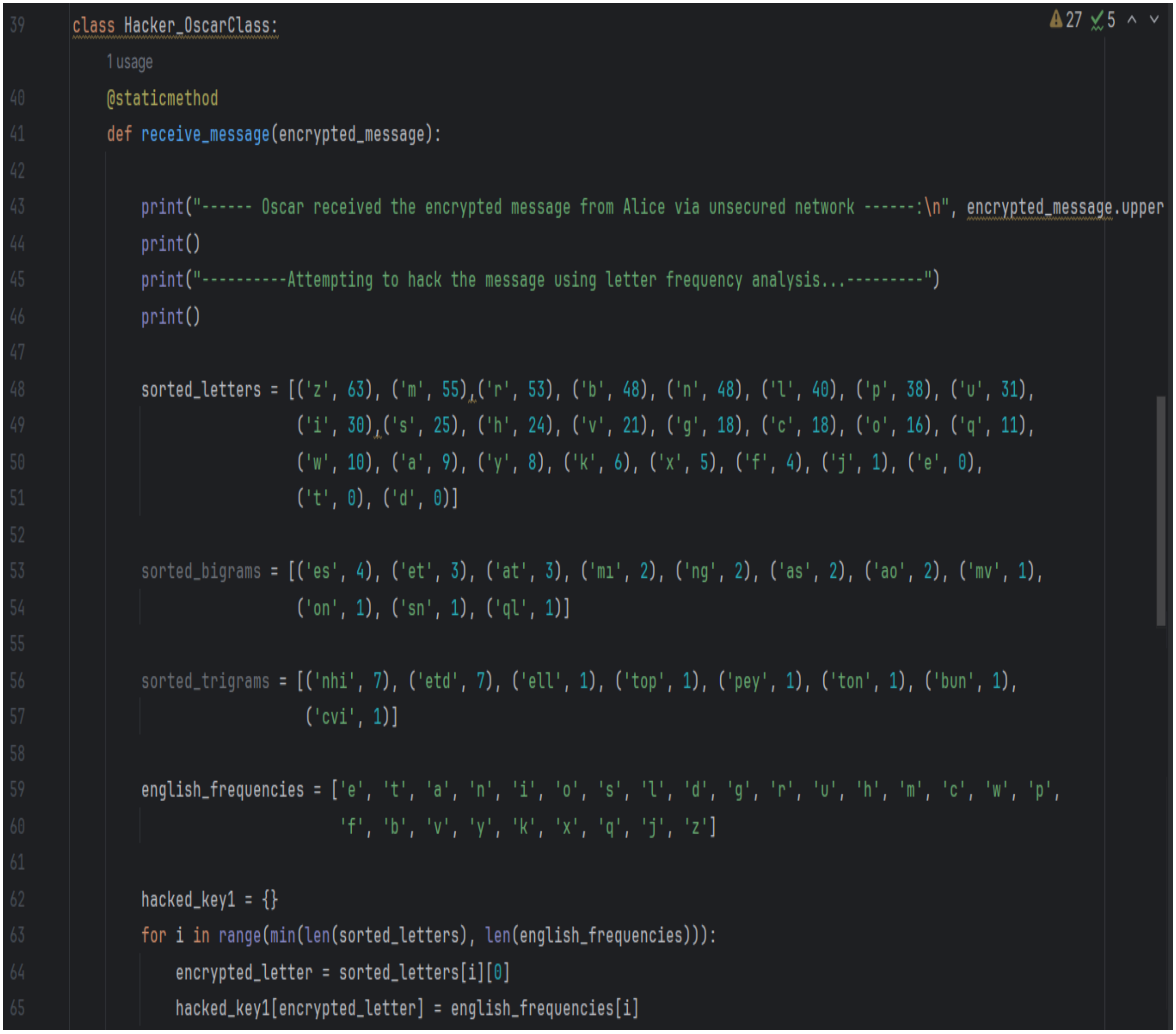
    • **If** the **char** key is among the values **(_Dkey.values())**, the key (original letter) corresponding to this value is found with the **next()** function and added to **Decrypted_message.**

    • **If char is not** among the values of the key or **char.isalpha()** returns false, the character is added directly to **Decrypted_message.**

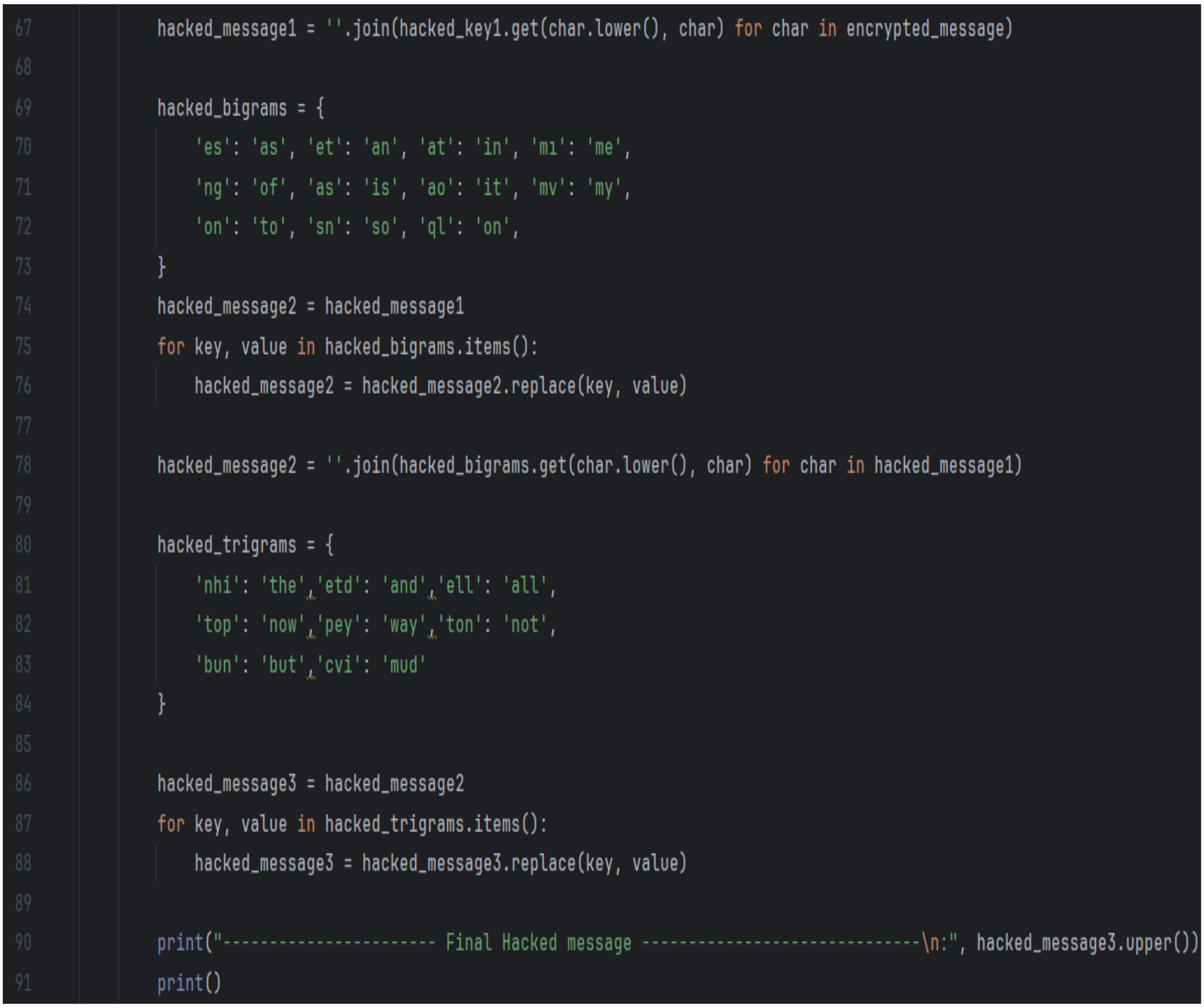
    • After the decryption process is finished, **Decrypted_message** is returned.

Hacker_Oscar_Class

This class simulates an attacker trying to decrypt the encrypted message using frequency analysis.



```
class Hacker_OscarClass:
    1 usage
    @staticmethod
    def receive_message(encrypted_message):

        print("------ Oscar received the encrypted message from Alice via unsecured network ------:\n", encrypted_message.upper
        print()
        print("---------Attempting to hack the message using letter frequency analysis...---------")
        print()

        sorted_letters = [('z', 63), ('m', 55),('r', 53), ('b', 48), ('n', 48), ('l', 40), ('p', 38), ('u', 31),
                          ('i', 30),('s', 25), ('h', 24), ('v', 21), ('g', 18), ('c', 18), ('o', 16), ('q', 11),
                          ('w', 10), ('a', 9), ('y', 8), ('k', 6), ('x', 5), ('f', 4), ('j', 1), ('e', 0),
                          ('t', 0), ('d', 0)]

        sorted_bigrams = [('es', 4), ('et', 3), ('at', 3), ('mı', 2), ('ng', 2), ('as', 2), ('ao', 2), ('mv', 1),
                          ('on', 1), ('sn', 1), ('ql', 1)]

        sorted_trigrams = [('nhi', 7), ('etd', 7), ('ell', 1), ('top', 1), ('pey', 1), ('ton', 1), ('bun', 1),
                           ('cvi', 1)]

        english_frequencies = ['e', 't', 'a', 'n', 'i', 'o', 's', 'l', 'd', 'g', 'r', 'u', 'h', 'm', 'c', 'w', 'p',
                               'f', 'b', 'v', 'y', 'k', 'x', 'q', 'j', 'z']

        hacked_key1 = {}
        for i in range(min(len(sorted_letters), len(english_frequencies))):
            encrypted_letter = sorted_letters[i][0]
            hacked_key1[encrypted_letter] = english_frequencies[i]
```



```
        hacked_message1 = ''.join(hacked_key1.get(char.lower(), char) for char in encrypted_message)

        hacked_bigrams = {
            'es': 'as', 'et': 'an', 'at': 'in', 'mı': 'me',
            'ng': 'of', 'as': 'is', 'ao': 'it', 'mv': 'my',
            'on': 'to', 'sn': 'so', 'ql': 'on',
        }
        hacked_message2 = hacked_message1
        for key, value in hacked_bigrams.items():
            hacked_message2 = hacked_message2.replace(key, value)

        hacked_message2 = ''.join(hacked_bigrams.get(char.lower(), char) for char in hacked_message1)

        hacked_trigrams = {
            'nhi': 'the','etd': 'and','ell': 'all',
            'top': 'now','pey': 'way','ton': 'not',
            'bun': 'but','cvi': 'mud'
        }

        hacked_message3 = hacked_message2
        for key, value in hacked_trigrams.items():
            hacked_message3 = hacked_message3.replace(key, value)

        print("--------------------- Final Hacked message ----------------------------\n:", hacked_message3.upper())
        print()
```

Here, I receive the encrypted message sent by **Alice** and write down the letters that I have manually analyzed in the incoming text, how many of each letter there are, and the double and triple letters one by one. First, I open a loop that changes the frequency letters one by one with the letters in the text I have solved with a for loop, so I can guess the words in the text a little. And here, I manually analyzed the text I extracted and then I was able to analyze **'Trigram'** and **'Bigram'**. I added them to my codes and made changes one by one. I transferred each analysis I made to the next text and finally printed out the final version in the **"Final Hacked message"** section. I'm pouring.

- **The receive_message method receives Alice's encrypted message and tries to decipher it by frequency analysis:**
- **sorted_letters, sorted_bigrams, sorted_trigrams:** Represents the frequencies of letter, two-letter and three-letter combinations in the encrypted message.
- **english_frequencies, bigram_frequencies, trigram_frequencies:** Represents the frequencies of letter, two-letter and three-letter combinations in the English language.

• **hacked_key1:** Generates a key based on letter frequency analysis. Matches scrambled letters to the most common letters in English.

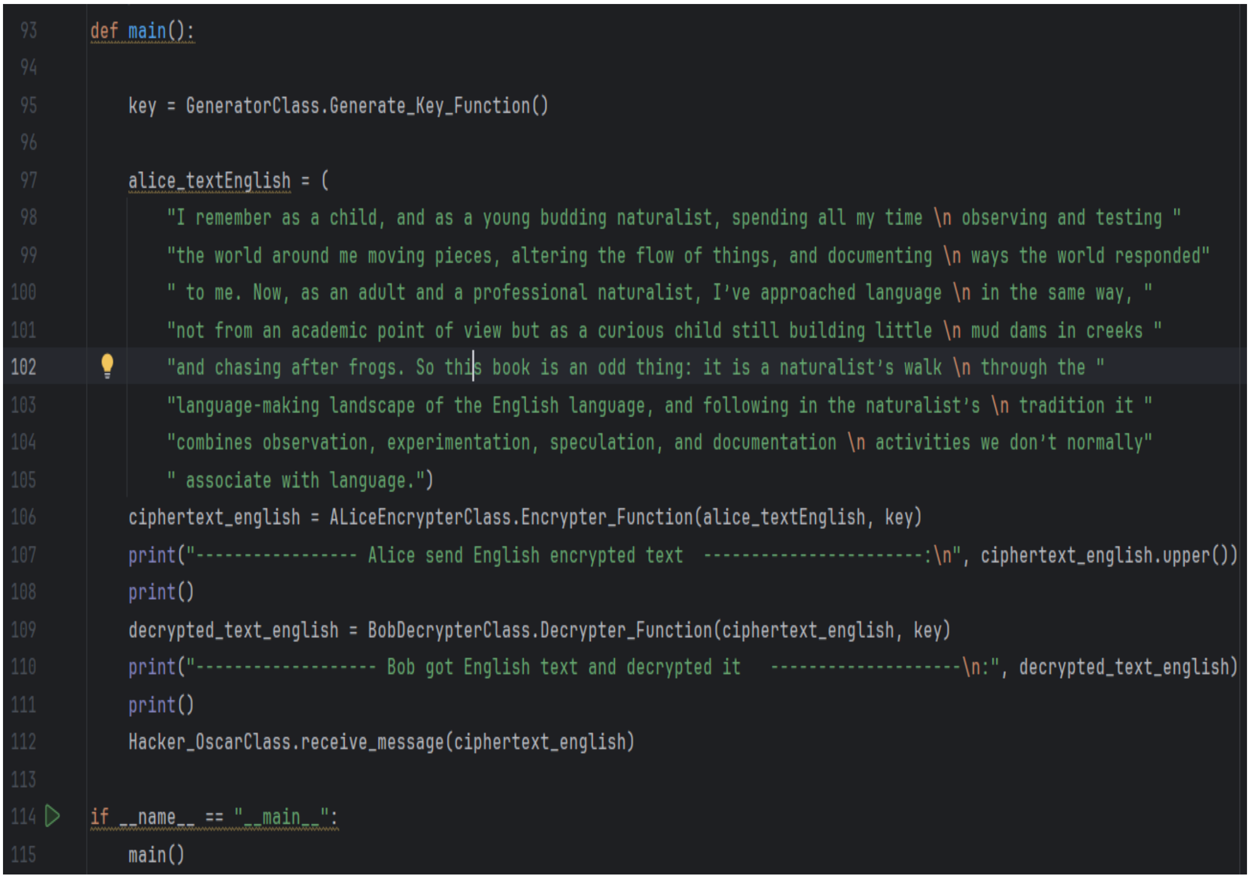
• **hacked_message1:** Decrypted message created using **hacked_key1.**

• **hacked_key2 and hacked_key3:** Performs additional analysis and modifications to **hacked_message1** to decipher two-letter and three-letter combinations**.**

• **hacked_message2 and hacked_message3:** Further decrypted messages resulting from these additional analyses.

Main Fonksiyonu

This function is the main function that performs all operations.



```python
def main():

    key = GeneratorClass.Generate_Key_Function()

    alice_textEnglish = (
        "I remember as a child, and as a young budding naturalist, spending all my time \n observing and testing "
        "the world around me moving pieces, altering the flow of things, and documenting \n ways the world responded"
        " to me. Now, as an adult and a professional naturalist, I’ve approached language \n in the same way, "
        "not from an academic point of view but as a curious child still building little \n mud dams in creeks "
        "and chasing after frogs. So this book is an odd thing: it is a naturalist’s walk \n through the "
        "language-making landscape of the English language, and following in the naturalist’s \n tradition it "
        "combines observation, experimentation, speculation, and documentation \n activities we don’t normally"
        " associate with language.")
    ciphertext_english = ALiceEncrypterClass.Encrypter_Function(alice_textEnglish, key)
    print("---------------- Alice send English encrypted text  ----------------------:\n", ciphertext_english.upper())
    print()
    decrypted_text_english = BobDecrypterClass.Decrypter_Function(ciphertext_english, key)
    print("------------------ Bob got English text and decrypted it   -------------------\n:", decrypted_text_english)
    print()
    Hacker_OscarClass.receive_message(ciphertext_english)

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

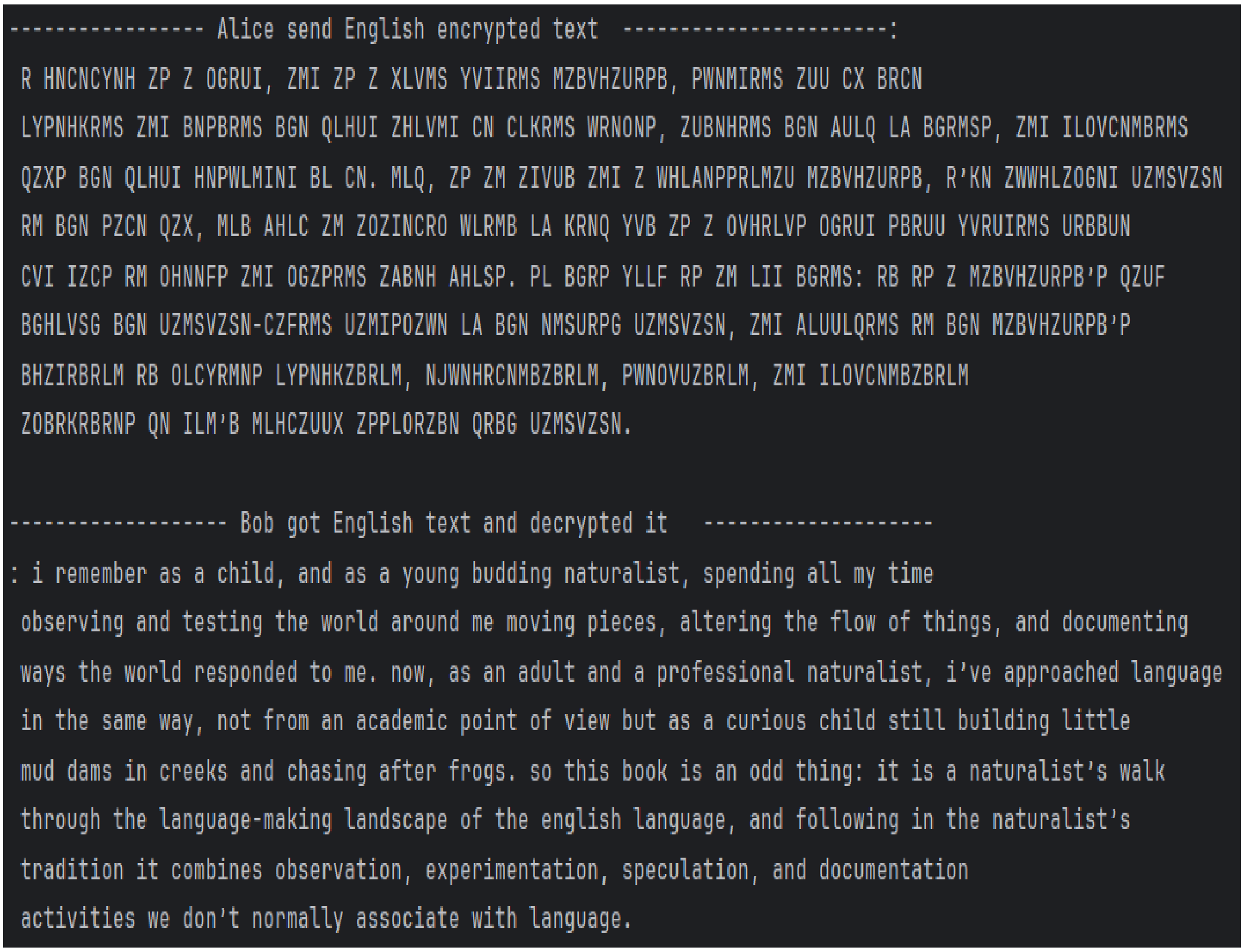
Here I now run my key and encryption function. I give the text to be encrypted. I print out the last encrypted text, the text Bob decrypted, and the cracked text. The results are below.

- **key**: It runs the Encryption and decryption key written above.
- **alice_textEnglish**: Alice's English text to be encrypted.
- **ciphertext_english**: Running the function that encrypts Alice's text.

- The encrypted text is printed on the screen.Bob, şifreli metni alır ve şifresini çözer.
- The deciphered text is printed on the screen.
- Oscar receives the ciphertext and tries to decipher it using various analysis methods. The hacked message is printed on the screen.

Output

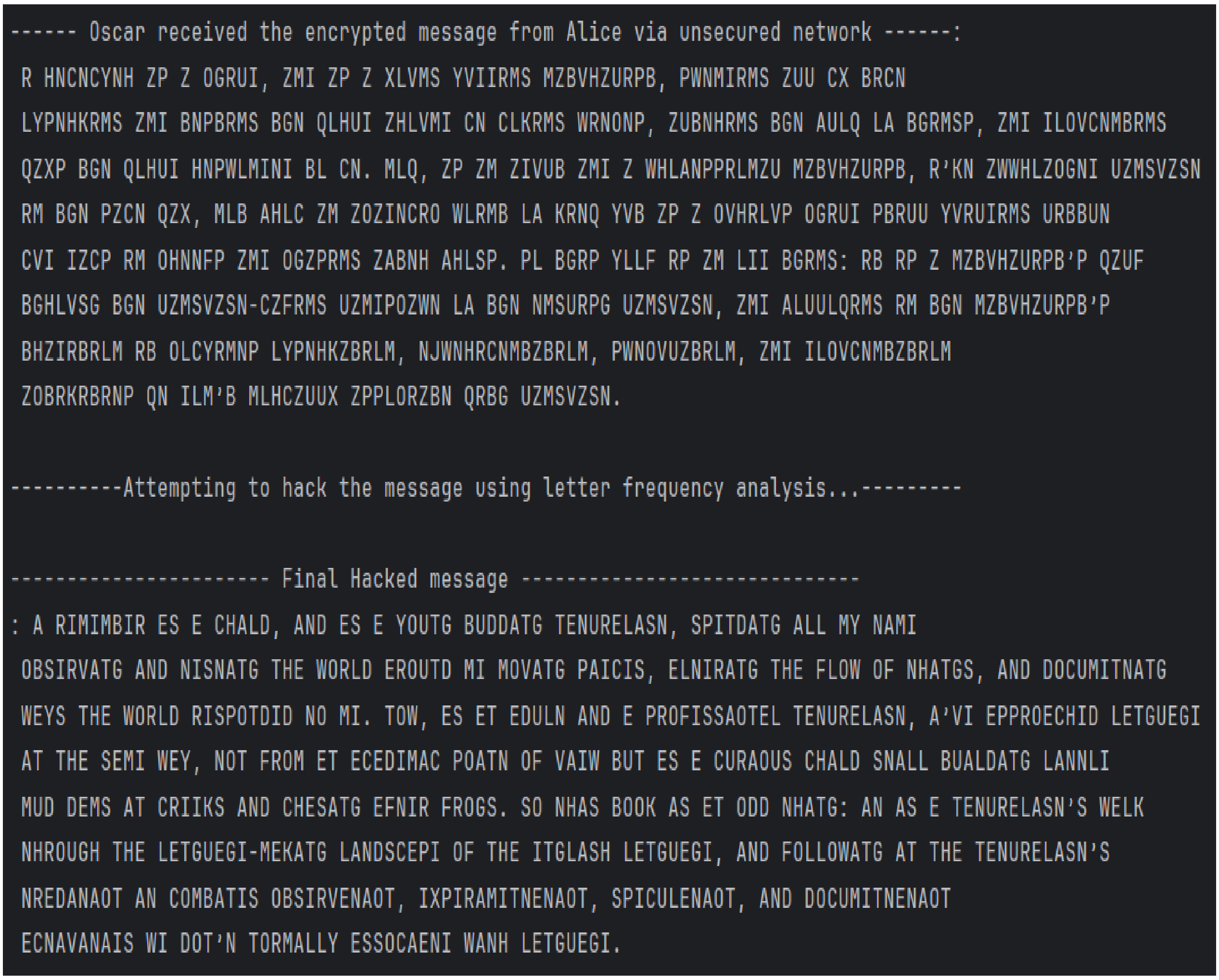
The outputs of the encryption and decryption operations are shown here.



```
----------------- Alice send English encrypted text  -----------------------:
 R HNCNCYNH ZP Z OGRUI, ZMI ZP Z XLVMS YVIIRMS MZBVHZURPB, PWNMIRMS ZUU CX BRCN
 LYPNHKRMS ZMI BNPBRMS BGN QLHUI ZHLVMI CN CLKRMS WRNONP, ZUBNHRMS BGN AULQ LA BGRMSP, ZMI ILOVCNMBRMS
 QZXP BGN QLHUI HNPWLMINI BL CN. MLQ, ZP ZM ZIVUB ZMI Z WHLANPPRLMZU MZBVHZURPB, R’KN ZWWHLZOGNI UZMSVZSN
 RM BGN PZCN QZX, MLB AHLC ZM ZOZINCRO WLRMB LA KRNQ YVB ZP Z OVHRLVP OGRUI PBRUU YVRUIRMS URBBUN
 CVI IZCP RM OHNNFP ZMI OGZPRMS ZABNH AHLSP. PL BGRP YLLF RP ZM LII BGRMS: RB RP Z MZBVHZURPB’P QZUF
 BGHLVSG BGN UZMSVZSN-CZFRMS UZMIPOZWN LA BGN NMSURPG UZMSVZSN, ZMI ALUULQRMS RM BGN MZBVHZURPB’P
 BHZIRBRLM RB OLCYRMNP LYPNHKZBRLM, NJWNHRCNMBZBRLM, PWNOVUZBRLM, ZMI ILOVCNMBZBRLM
 ZOBRKRBRNP QN ILM’B MLHCZUUX ZPPLORZBN QRBG UZMSVZSN.


------------------- Bob got English text and decrypted it   --------------------
: i remember as a child, and as a young budding naturalist, spending all my time
 observing and testing the world around me moving pieces, altering the flow of things, and documenting
 ways the world responded to me. now, as an adult and a professional naturalist, i’ve approached language
 in the same way, not from an academic point of view but as a curious child still building little
 mud dams in creeks and chasing after frogs. so this book is an odd thing: it is a naturalist’s walk
 through the language-making landscape of the english language, and following in the naturalist’s
 tradition it combines observation, experimentation, speculation, and documentation
 activities we don’t normally associate with language.
```

The outputs of the encryption and decryption operations are shown here.

Here, Oscar's password cracking process is shown by analysis.



```
------ Oscar received the encrypted message from Alice via unsecured network ------:
 R HNCNCYNH ZP Z OGRUI, ZMI ZP Z XLVMS YVIIRMS MZBVHZURPB, PWNMIRMS ZUU CX BRCN
 LYPNHKRMS ZMI BNPBRMS BGN QLHUI ZHLVMI CN CLKRMS WRNONP, ZUBNHRMS BGN AULQ LA BGRMSP, ZMI ILOVCNMBRMS
 QZXP BGN QLHUI HNPWLMINI BL CN. MLQ, ZP ZM ZIVUB ZMI Z WHLANPPRLMZU MZBVHZURPB, R’KN ZWWHLZOGNI UZMSVZSN
 RM BGN PZCN QZX, MLB AHLC ZM ZOZINCRO WLRMB LA KRNQ YVB ZP Z OVHRLVP OGRUI PBRUU YVRUIRMS URBBUN
 CVI IZCP RM OHNNFP ZMI OGZPRMS ZABNH AHLSP. PL BGRP YLLF RP ZM LII BGRMS: RB RP Z MZBVHZURPB’P QZUF
 BGHLVSG BGN UZMSVZSN-CZFRMS UZMIPOZWN LA BGN NMSURPG UZMSVZSN, ZMI ALUULQRMS RM BGN MZBVHZURPB’P
 BHZIRBRLM RB OLCYRMNP LYPNHKZBRLM, NJWNHRCNMBZBRLM, PWNOVUZBRLM, ZMI ILOVCNMBZBRLM
 ZOBRKRBRNP QN ILM’B MLHCZUUX ZPPLORZBN QRBG UZMSVZSN.


----------Attempting to hack the message using letter frequency analysis...---------


---------------------- Final Hacked message ------------------------------
: A RIMIMBIR ES E CHALD, AND ES E YOUTG BUDDATG TENURELASN, SPITDATG ALL MY NAMI
 OBSIRVATG AND NISNATG THE WORLD EROUTD MI MOVATG PAICIS, ELNIRATG THE FLOW OF NHATGS, AND DOCUMITNATG
 WEYS THE WORLD RISPOTDID NO MI. TOW, ES ET EDULN AND E PROFISSAOTEL TENURELASN, A’VI EPPROECHID LETGUEGI
 AT THE SEMI WEY, NOT FROM ET ECEDIMAC POATN OF VAIW BUT ES E CURAOUS CHALD SNALL BUALDATG LANNLI
 MUD DEMS AT CRIIKS AND CHESATG EFNIR FROGS. SO NHAS BOOK AS ET ODD NHATG: AN AS E TENURELASN’S WELK
 NHROUGH THE LETGUEGI-MEKATG LANDSCEPI OF THE ITGLASH LETGUEGI, AND FOLLOWATG AT THE TENURELASN’S
 NREDANAOT AN COMBATIS OBSIRVENAOT, IXPIRAMITNENAOT, SPICULENAOT, AND DOCUMITNENAOT
 ECNAVANAIS WI DOT’N TORMALLY ESSOCAENI WANH LETGUEGI.
```

Here, Oscar's password cracking process is shown by analysis.

Here the encrypted message is read at an understandable level. The remaining letters can be fully solved by manual testing and guessing. And it can be added to the coding. But since I was proceeding according to the scenario, I did not want to go beyond the desired task and realism.

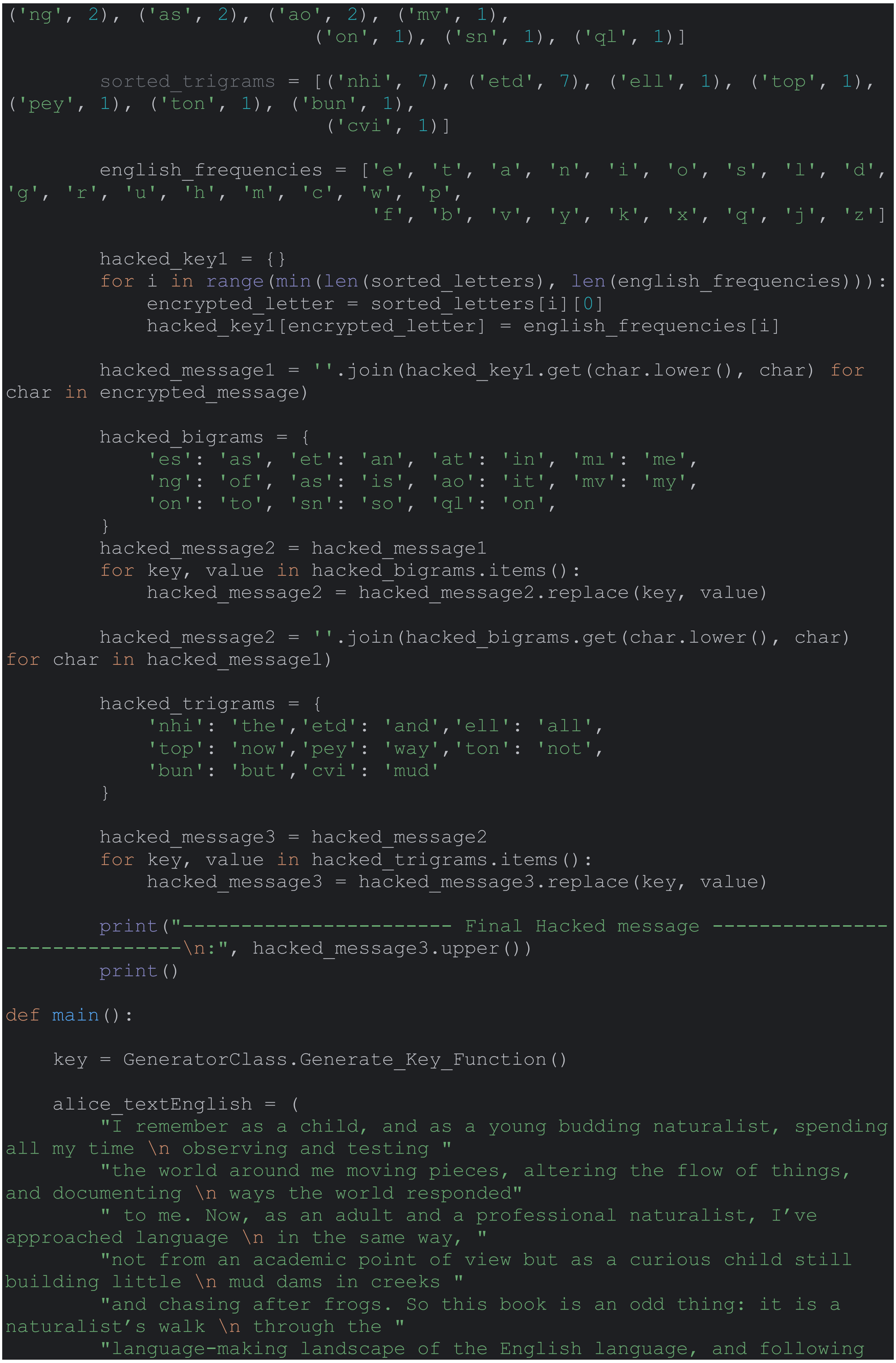
Summary

This code contains a simulation that demonstrates the encryption and decryption processes, as well as how an attacker (Oscar) might attempt to decipher the message

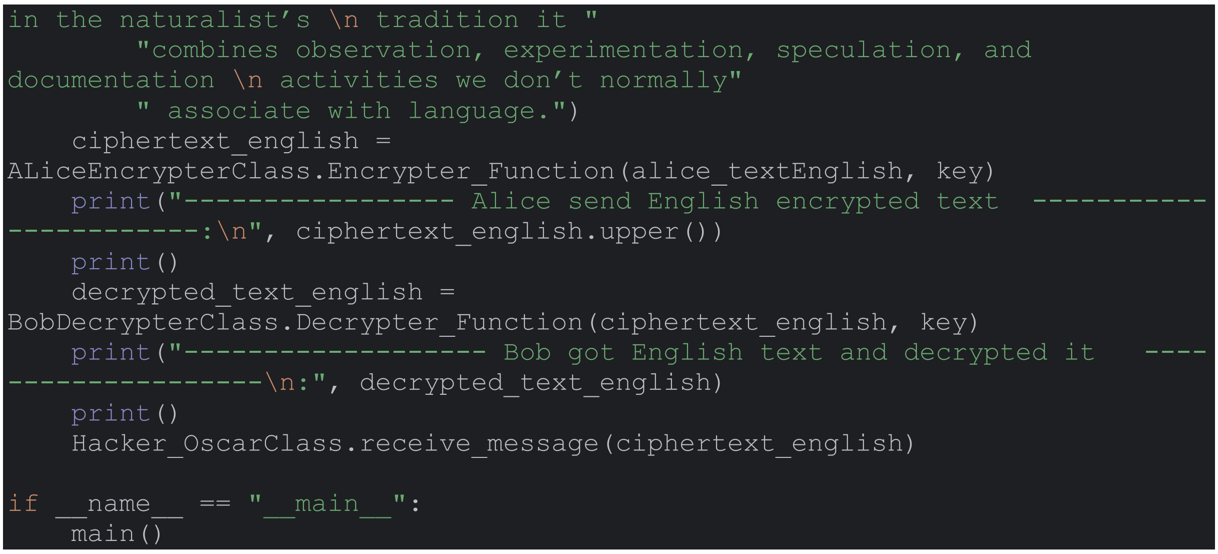
using methods such as frequency analysis. It is shown how a message is encrypted, decrypted and attempted to be cracked through the characters Alice, Bob and Oscar.

MY CODES

```
class GeneratorClass:
    @staticmethod
    def Generate_Key_Function():
        open_alphabet =   'abcdefghijklmnopqrstuvwxyz'
        cipher_alphabet = 'zyoinasgrdfucmlwthpbvkqjxe'
        key_mapping = dict(zip(open_alphabet, cipher_alphabet))
        return key_mapping

class ALiceEncrypterClass:
    @staticmethod
    def Encrypter_Function(_Emessage, _Ekey):
        Encrypted_message = ''
        for char in _Emessage:
            if char.isalpha():
                char = char.lower()
                if char in _Ekey:
                    Encrypted_message += _Ekey[char]
                else:
                    Encrypted_message += char
            else:
                Encrypted_message += char
        return Encrypted_message

class BobDecrypterClass:
    @staticmethod
    def Decrypter_Function(_Dciphertext, _Dkey):
        Decrypted_message = ''
        for char in _Dciphertext:
            if char.isalpha():
                char = char.lower()
                if char in _Dkey.values():
                    Decrypted_message += next(k for k, v in _Dkey.items()
if v == char)
                else:
                    Decrypted_message += char
            else:
                Decrypted_message += char
        return Decrypted_message

class Hacker_OscarClass:
    @staticmethod
    def receive_message(encrypted_message):

        print("------ Oscar received the encrypted message from Alice via
unsecured network ------:\n", encrypted_message.upper())
        print()
        print("----------Attempting to hack the message using letter
frequency analysis...---------")
        print()

        sorted_letters = [('z', 63), ('m', 55),('r', 53), ('b', 48), ('n',
48), ('l', 40), ('p', 38), ('u', 31),
                          ('i', 30),('s', 25), ('h', 24), ('v', 21), ('g',
18), ('c', 18), ('o', 16), ('q', 11),
                          ('w', 10), ('a', 9), ('y', 8), ('k', 6), ('x',
5), ('f', 4), ('j', 1), ('e', 0),
                          ('t', 0), ('d', 0)]

        sorted_bigrams = [('es', 4), ('et', 3), ('at', 3), ('mı', 2),
```



```
('ng', 2), ('as', 2), ('ao', 2), ('mv', 1),
                          ('on', 1), ('sn', 1), ('ql', 1)]

        sorted_trigrams = [('nhi', 7), ('etd', 7), ('ell', 1), ('top', 1),
('pey', 1), ('ton', 1), ('bun', 1),
                           ('cvi', 1)]

        english_frequencies = ['e', 't', 'a', 'n', 'i', 'o', 's', 'l', 'd',
'g', 'r', 'u', 'h', 'm', 'c', 'w', 'p',
                               'f', 'b', 'v', 'y', 'k', 'x', 'q', 'j', 'z']

        hacked_key1 = {}
        for i in range(min(len(sorted_letters), len(english_frequencies))):
            encrypted_letter = sorted_letters[i][0]
            hacked_key1[encrypted_letter] = english_frequencies[i]

        hacked_message1 = ''.join(hacked_key1.get(char.lower(), char) for
char in encrypted_message)

        hacked_bigrams = {
            'es': 'as', 'et': 'an', 'at': 'in', 'mı': 'me',
            'ng': 'of', 'as': 'is', 'ao': 'it', 'mv': 'my',
            'on': 'to', 'sn': 'so', 'ql': 'on',
        }
        hacked_message2 = hacked_message1
        for key, value in hacked_bigrams.items():
            hacked_message2 = hacked_message2.replace(key, value)

        hacked_message2 = ''.join(hacked_bigrams.get(char.lower(), char)
for char in hacked_message1)

        hacked_trigrams = {
            'nhi': 'the','etd': 'and','ell': 'all',
            'top': 'now','pey': 'way','ton': 'not',
            'bun': 'but','cvi': 'mud'
        }

        hacked_message3 = hacked_message2
        for key, value in hacked_trigrams.items():
            hacked_message3 = hacked_message3.replace(key, value)

        print("----------------------- Final Hacked message ---------------
---------------\n:", hacked_message3.upper())
        print()

def main():

    key = GeneratorClass.Generate_Key_Function()

    alice_textEnglish = (
        "I remember as a child, and as a young budding naturalist, spending
all my time \n observing and testing "
        "the world around me moving pieces, altering the flow of things,
and documenting \n ways the world responded"
        " to me. Now, as an adult and a professional naturalist, I’ve
approached language \n in the same way, "
        "not from an academic point of view but as a curious child still
building little \n mud dams in creeks "
        "and chasing after frogs. So this book is an odd thing: it is a
naturalist’s walk \n through the "
        "language-making landscape of the English language, and following
```



```
in the naturalist’s \n tradition it "
        "combines observation, experimentation, speculation, and
documentation \n activities we don’t normally"
        " associate with language.")
    ciphertext_english =
ALiceEncrypterClass.Encrypter_Function(alice_textEnglish, key)
    print("----------------- Alice send English encrypted text  -----------
------------:\n", ciphertext_english.upper())
    print()
    decrypted_text_english =
BobDecrypterClass.Decrypter_Function(ciphertext_english, key)
    print("------------------- Bob got English text and decrypted it   ----
----------------\n:", decrypted_text_english)
    print()
    Hacker_OscarClass.receive_message(ciphertext_english)

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