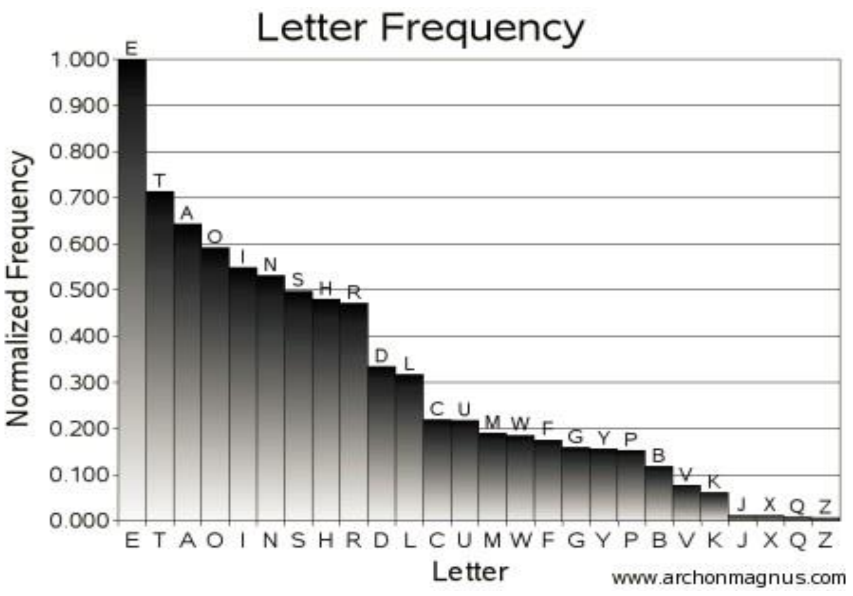
Lab 2 Report: Breaking without Brute Force

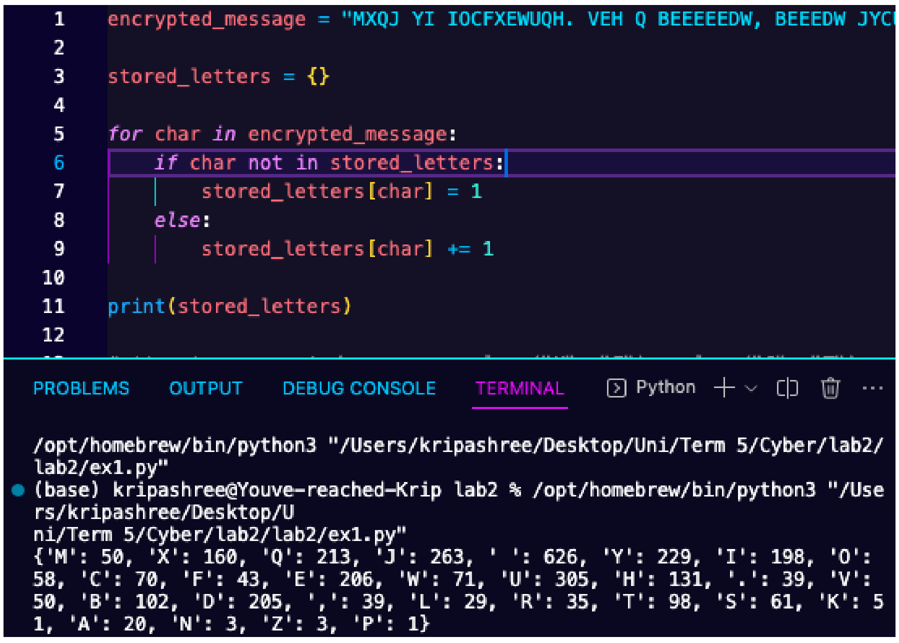
Name: Kripashree Suryaprakash
Student ID: 1006507

Part I: Substitution Cipher

I used the frequency analysis to decrypt the cipher text, according to the statistics presented in the graph below.



I proceeded to find the frequency of alphabets in the given cipher text. This would give me a better idea of matching the respective alphabets in my cipher text to the frequency of alphabet occurrences in the English language. Spaces and punctuation marks were preserved during decryption.



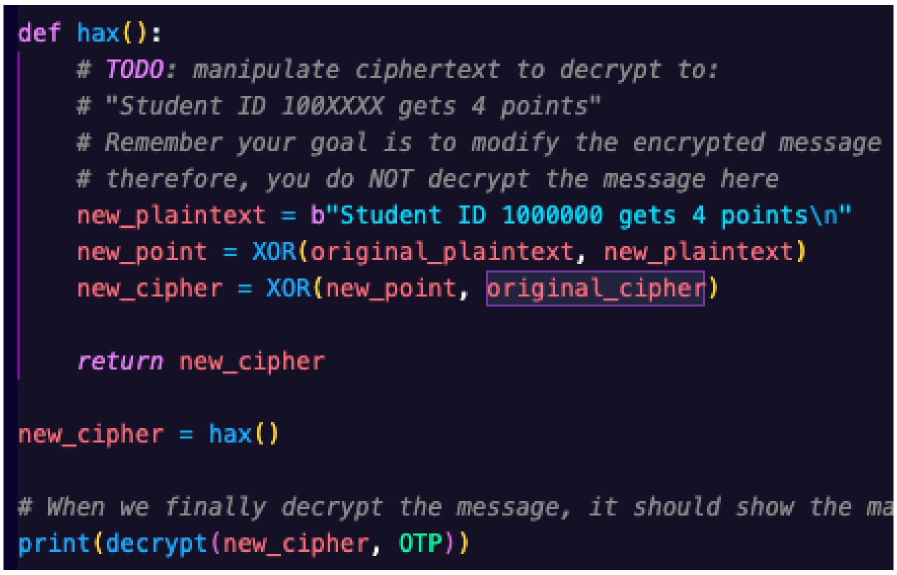
```python
encrypted_message = "MXQJ YI IOCFXEWUQH. VEH Q BEEEEEDW, BEEEDW JYC

stored_letters = {}

for char in encrypted_message:
    if char not in stored_letters:
        stored_letters[char] = 1
    else:
        stored_letters[char] += 1

print(stored_letters)

```

```
/opt/homebrew/bin/python3 "/Users/kripashree/Desktop/Uni/Term 5/Cyber/lab2/
lab2/ex1.py"
● (base) kripashree@Youve-reached-Krip lab2 % /opt/homebrew/bin/python3 "/Use
rs/kripashree/Desktop/U
ni/Term 5/Cyber/lab2/lab2/ex1.py"
{'M': 50, 'X': 160, 'Q': 213, 'J': 263, ' ': 626, 'Y': 229, 'I': 198, 'O':
58, 'C': 70, 'F': 43, 'E': 206, 'W': 71, 'U': 305, 'H': 131, '.': 39, 'V':
50, 'B': 102, 'D': 205, ',': 39, 'L': 29, 'R': 35, 'T': 98, 'S': 61, 'K': 5
1, 'A': 20, 'N': 3, 'Z': 3, 'P': 1}
```

After many trial and error testing rounds, I decrypted the cipher text using the Python's string *'.replace'* function. I could identify the corresponding alphabets and their ciphers as I recognized more and more familiar word blocks. The most common words that appeared in the first two rounds of trial and error were "I", "are", "can" and so on.

Part II: Compromising OTP Integrity

The aim of this exercise was to change the decrypted plain text response to say we have gotten 4 points, without decrypting it.



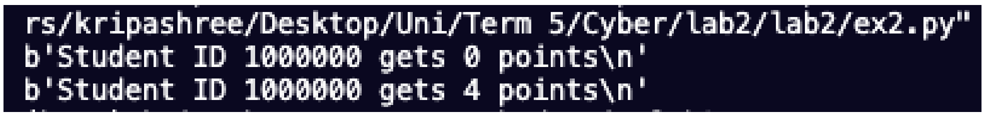
```
def hax():
    # TODO: manipulate ciphertext to decrypt to:
    # "Student ID 100XXXX gets 4 points"
    # Remember your goal is to modify the encrypted message
    # therefore, you do NOT decrypt the message here
    new_plaintext = b"Student ID 1000000 gets 4 points\n"
    new_point = XOR(original_plaintext, new_plaintext)
    new_cipher = XOR(new_point, original_cipher)

    return new_cipher

new_cipher = hax()

# When we finally decrypt the message, it should show the ma
print(decrypt(new_cipher, OTP))
```

I created the XOR formulae for changing the point from "0" to "4", as well as for changing the cipher text to print the new cipher text with the changed point value.

A' = A ⊕ OTP
B' = B ⊕ OTP
where, A is the original text, B is the plaintext and A' is the original cipher, B' is the new cipher.

A ⊕ OTP ⊕ A ⊕ B = A' ⊕ A ⊕ B
A ⊕ A = 0 is nulled out based on the proof X ⊕ X = 0. Hence, OTP ⊕ B = B', where B' is the new cipher.

As a result, decrypting the new cipher by implementing the XOR functions in the *hax()* function gives us the output with "4" points in place of "0" points.



```
rs/kripashree/Desktop/Uni/Term 5/Cyber/lab2/lab2/ex2.py"
b'Student ID 1000000 gets 0 points\n'
b'Student ID 1000000 gets 4 points\n'
```