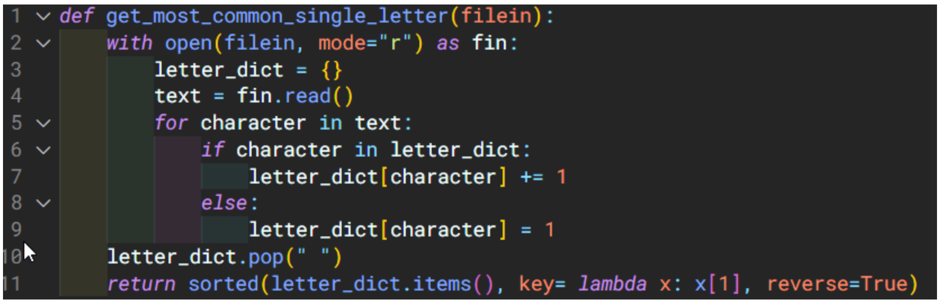
Exercise 1:

- For exercise 1, I opted to use the approach to find the most common letter, bigram and trigram and map it to the most common English counterparts as shown in the Wikipedia link provided

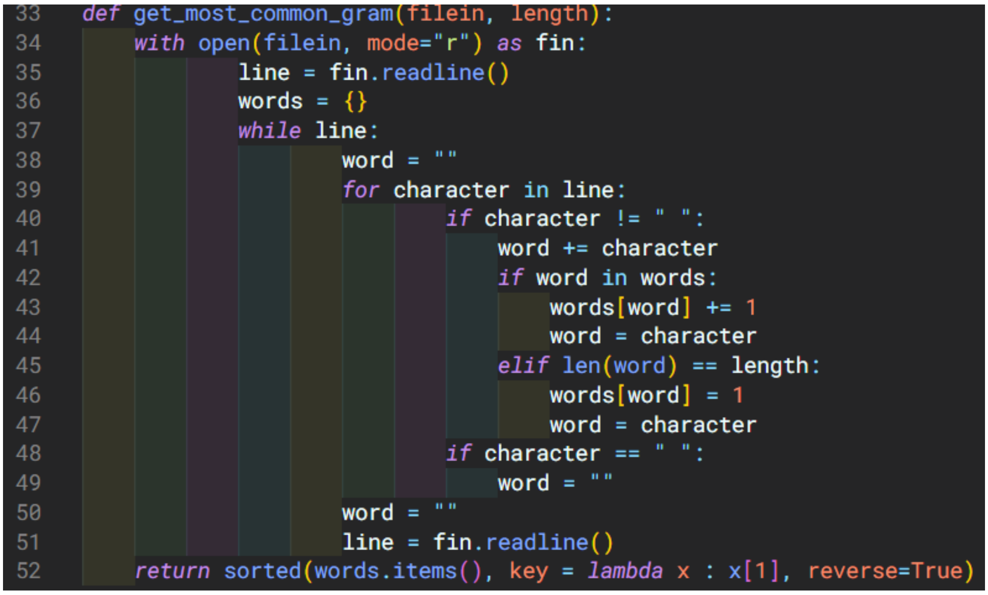
Get Most Common _____

- The `get_most_common_single_letter` function returns a list of all the letters and their frequencies in the format (<Letter>, <Frequency>)



```python
def get_most_common_single_letter(filein):
    with open(filein, mode="r") as fin:
        letter_dict = {}
        text = fin.read()
        for character in text:
            if character in letter_dict:
                letter_dict[character] += 1
            else:
                letter_dict[character] = 1
    letter_dict.pop(" ")
    return sorted(letter_dict.items(), key= lambda x: x[1], reverse=True)
```

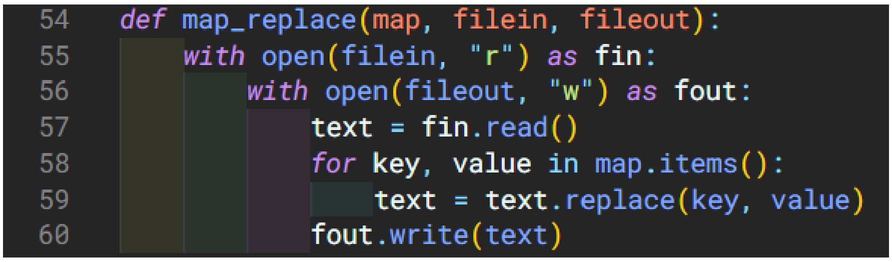
- The `get_most_common_gram` function takes in a parameter *length* where *length* = 2 indicates a bigram and *length* = 3 indicates a trigram. For simplicity, I will denote possible combinations of letters, regardless of length, as “gram” for generic use. This function returns a list of all the possible grams and their frequencies in the format (<Gram>, <Frequency>)



```python
def get_most_common_gram(filein, length):
    with open(filein, mode="r") as fin:
            line = fin.readline()
            words = {}
            while line:
                    word = ""
                    for character in line:
                            if character != " ":
                                word += character
                                if word in words:
                                    words[word] += 1
                                    word = character
                                elif len(word) == length:
                                    words[word] = 1
                                    word = character
                            if character == " ":
                                word = ""
                    word = ""
                    line = fin.readline()
    return sorted(words.items(), key = lambda x : x[1], reverse=True)
```

Replace Letters

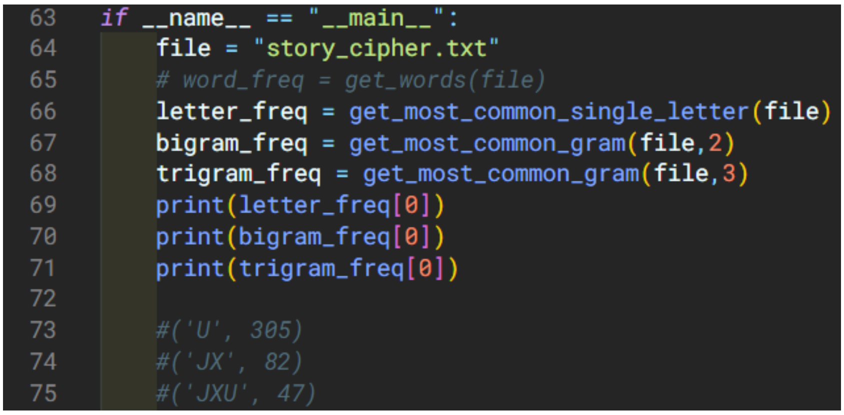
- The `map_replace` function takes in a parameter *map*. It will read the input file, replace the alphabets based on the *map* and write it to an output file.



```python
def map_replace(map, filein, fileout):
    with open(filein, "r") as fin:
        with open(fileout, "w") as fout:
            text = fin.read()
            for key, value in map.items():
                text = text.replace(key, value)
            fout.write(text)
```

Main Function

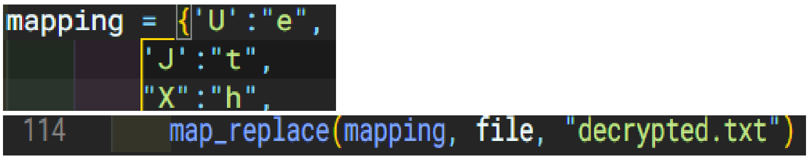
- Under the main function, I first printed out the most used letter, bigram and trigram using the functions explained above.



```python
if __name__ == "__main__":
    file = "story_cipher.txt"
    # word_freq = get_words(file)
    letter_freq = get_most_common_single_letter(file)
    bigram_freq = get_most_common_gram(file,2)
    trigram_freq = get_most_common_gram(file,3)
    print(letter_freq[0])
    print(bigram_freq[0])
    print(trigram_freq[0])

    #('U', 305)
    #('JX', 82)
    #('JXU', 47)
```

- Under the Wikipedia page provided, “e is the most common letter in the English language, th is the most common bigram, and the is the most common trigram.”
- In this case, (U, JX, JXU) corresponded very nicely to (e, th, the) and thus, they were the first few inclusions in my map



```python
mapping = {'U':"e",
          'J':"t",
          "X":"h",
```

```python
    map_replace(mapping, file, "decrypted.txt")
```

- After this, I ran the `map_replace` function to try and find clues for other letters that appear in common English words
- Similar process and iterations were made as shown in the table below (Capital letters indicate the encrypted alphabets and lower case letters indicate the decrypted alphabets)

Iteration	Comment	New Maps
1	can see "thQt", which implies that Q = a	Q = a
2	Short words provide useful clues. One-letter words are either a or i. I can see that Y also appears alone, and since Q = a, Y = i	Y = i
3	I can see this combination "it iI", can assume that I = s	I = s
4	I can see TiT, can assume it that T = d	T = d

5	"i did DEt KDdeHstaDd Mhat it is" looks like "I did not understand what it is"	D = n,
		E = o,
		K = u,
		H = r,
		M = w

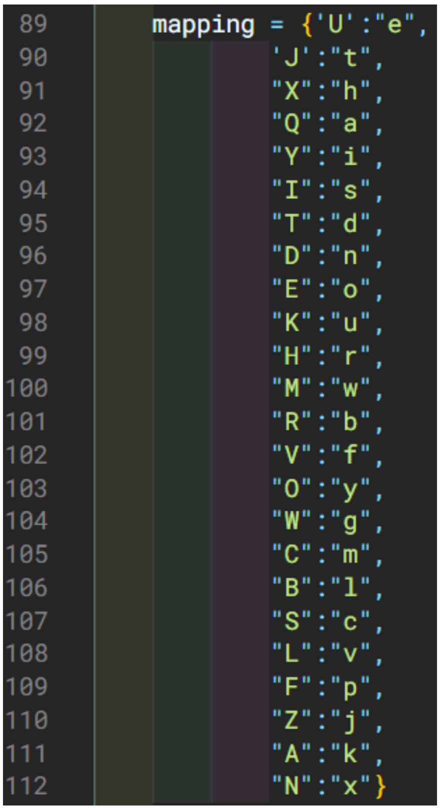
6	“Reen a satisVOinW one throuWhout” looks like “been a satisfying one throughout”	R = b,
		V = f,
		O = y,
		W = g

7	CyseBf in this franShise looks like “myself in this franchise”	C = m,
		B = l,
		S = c

8	“haLe neLer” looks like “have never”	L = v

9	“imFortantly” looks like “importantly”
	“maZor” looks like “major”
	“Anow” looks like “know”
	eNFect” looks like “expect
		F = p,
		Z = j,
		A = k,
		N = x

- After going through all these iterations, this is the final mapping of the letters based on frequency analysis and observation

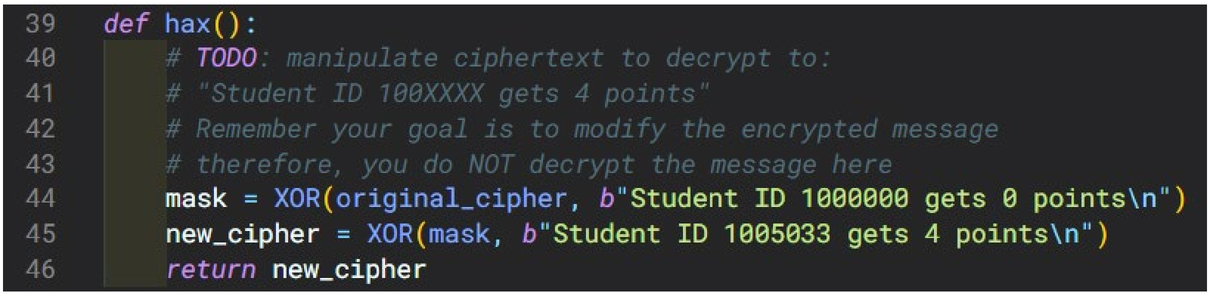


```
    mapping = {'U':"e",
               'J':"t",
               "X":"h",
               "Q":"a",
               "Y":"i",
               "I":"s",
               "T":"d",
               "D":"n",
               "E":"o",
               "K":"u",
               "H":"r",
               "M":"w",
               "R":"b",
               "V":"f",
               "O":"y",
               "W":"g",
               "C":"m",
               "B":"l",
               "S":"c",
               "L":"v",
               "F":"p",
               "Z":"j",
               "A":"k",
               "N":"x"}
```

- Looking through the decrypted text file, I can see that there are no more capital letters in the text file, which means that all the encrypted alphabets are included in the map and decrypted.

Exercise 2:

- Using the commutative property of XOR(^), it can be seen that 100 ^ OTP ^ 100 ^ 999 = OTP ^ 999
- With this property, given that I know the final message is supposed to be “Student ID 100XXXX gets 4 points", while the original message is "Student ID 1000000 gets 0 points"
  - I first create a mask by using the formula mask = (original_cipher ^ original plaintext)
    - This mask is the binary version of the OTP due to the commutative property of ^
  - I then create the new_cipher by using the formula new_cipher = (mask ^ “Student ID 100XXXX gets 4 points")
  - The steps denoted are written in the hax function



```
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
    mask = XOR(original_cipher, b"Student ID 1000000 gets 0 points\n")
    new_cipher = XOR(mask, b"Student ID 1005033 gets 4 points\n")
    return new_cipher
```

- Finally, the new_cipher will decrypt to “Student ID 100XXXX gets 4 points"