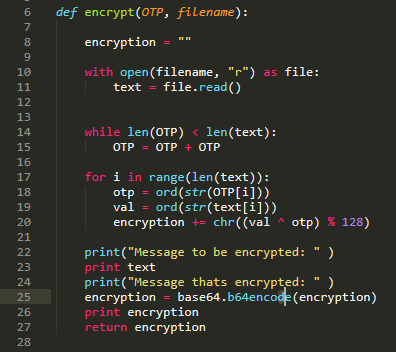
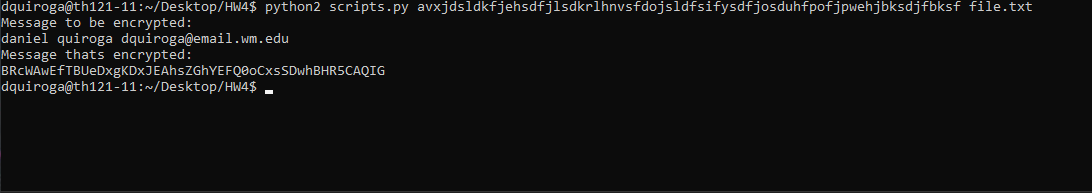
Task 1

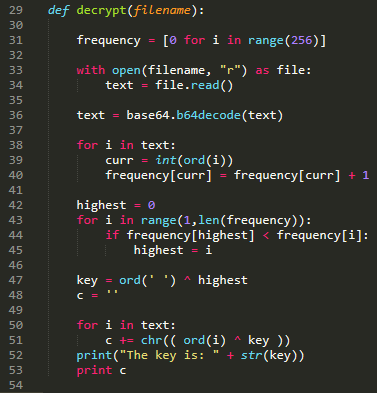
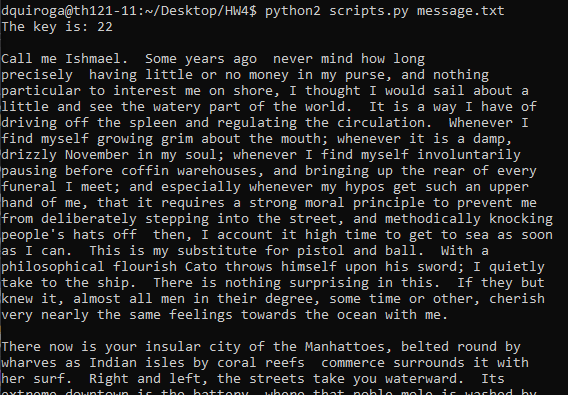
1. My code first checks to see the OTP has the proper length to encrypt the message and then goes through each index and performs an addition and mod of the ordinal alphabet. Once converted to a corresponding character it adds it to the output string and then at the very end prints out the original message and the encrypted message. I also added an encoding of base64 to provide a better output. Attached is a screen shot of both the code and the output generated by the code:





1. This would provide some sort of security but not secure (due to the Two-Time Pad Problem) since the message is being delivered through a public channel and many students are using the same pad to encrypt their messages this allows an attacker an abundance of encrypted text with the same pad and hence they are able to being to extract which characters may refer to what and eventually break what the encryption is. I would improve security by having the pad be different for every student in the class, this would decrease the probably that an attacker breaks the encryption and read the contents of messages.

Task 2

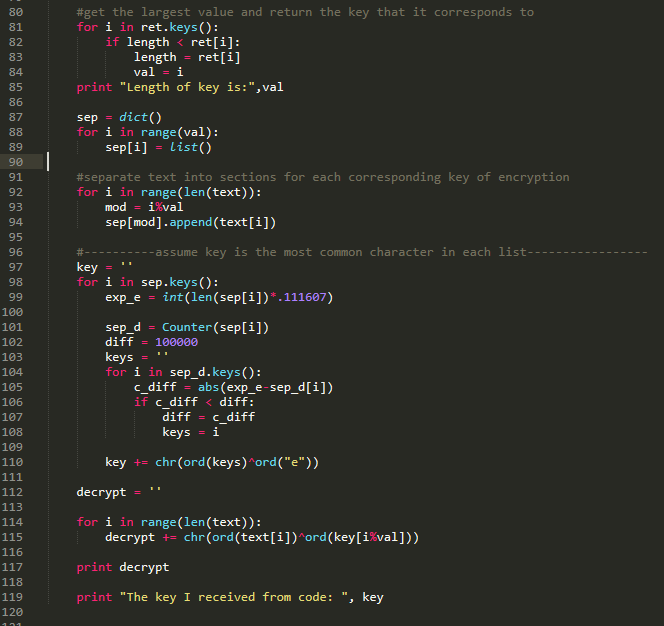
1. Here we can see two ways that would be able to break this encryption.
   1. A brute-force attack would try each and every combination of the rotation (so all 255) and eventually get the right rotation that was chosen.
   2. A frequency analysis, look which character appears the most and use it to match it back to what a regular frequency of characters is typically and eventually one of your options would give you the rotation.
2. I used the frequency method to find what the key was used in the encoded message. Using the example program provided, I first decoded the text since it was stated that message was encoded in base64. Afterwards, I updated the corresponding ordinal index based on the frequency that a value would appear. Knowing that “ ” should be the character to appear most I found which index contained the highest value and did a reverse calculation to get the key that was used and then reversed the encryption method by doing the XOR to get the original key then just go through each character and XOR to get the original character. The key I got from my method was 22.

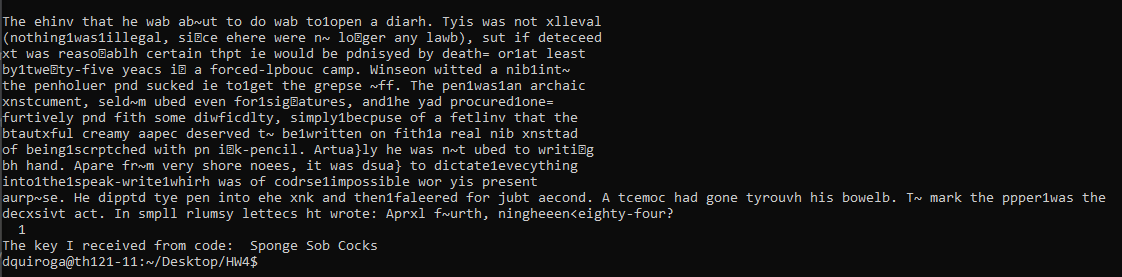
Task 3:

1. When a repeated sequence of symbols is found in the ciphertext, this means that there is a good possibility that these parts were encrypted with the same key parts. By seeing how far apart the repeated sequence is, we get an idea of how long the key is (i.e. by the factors of the distance). Once we decide on the length, we perform a statistical analysis with n groups.
2. I first converted all characters in the message into ordinal values and placed them into a list. I then created a helper function to find sequences in a greedy methodology. In the function repeition I first will iterate through each index in the list of ordinals and then the j variable references the number of spaces that a repeition could be apart (which I gave an arbitrary number 20) and then x is the sequence that will match, hence a sequence of 2-20 since that’s how long our key could be. Once returned I place all the values into a dictionary and then iterate through the dictionary to see which value had come up the most and that is the length of the key. (in my run, 16 won by over 100 hence was a really easy check to see if this was reasonable)

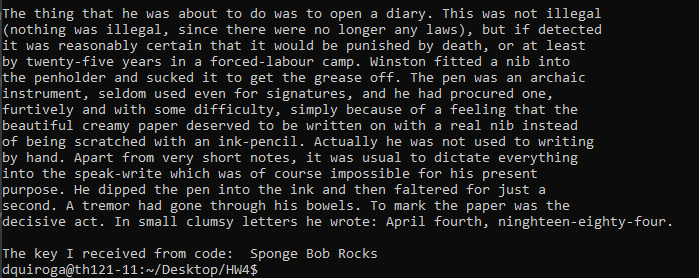


1. The following portion of my code gets me the key by checking to see which character appears the most in the smaller groups by indices. I get the key that appears the same as the expected number of times “e” should appear in the length of that list. Hence statistically we would assume that the original text being encrypted was an e and hence get the corresponding key value for that index. The output of the key is “Sponge Sob Cocks” but I can fairly easily tell that the correct key should be “Sponge Bob Rocks”.



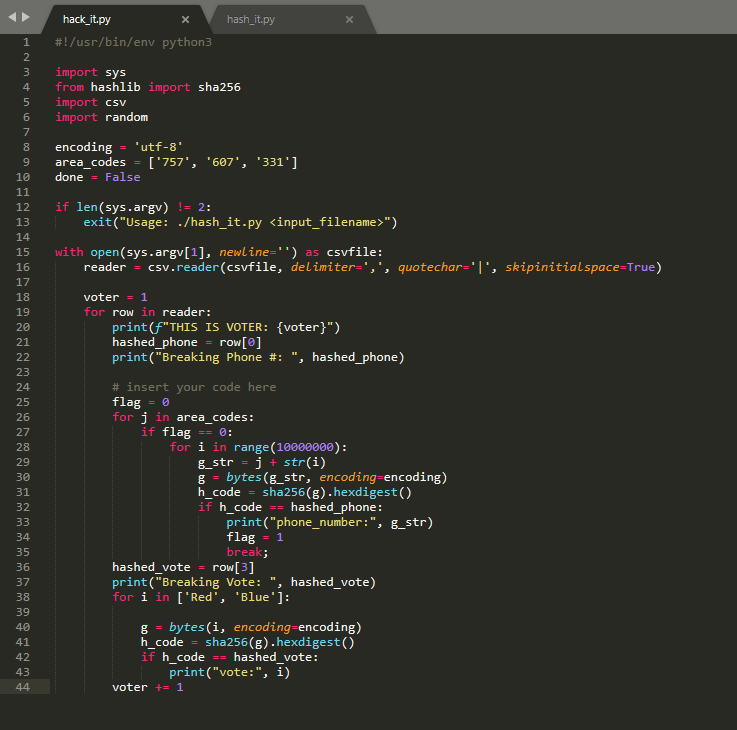
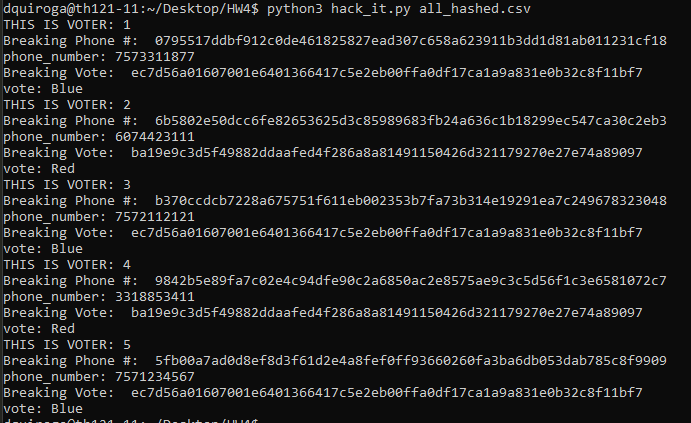


But when I hard code the corrected key I get the following output:



Which looks like a completely decrypted text.

Task 4

Below is the file I used to get the hashed values and the output after running it. I did a brute force iteration of all numbers between 0, 10000000 and iterated through the given list of area codes to save time. I would hash each and then check to see if the hash value is the same as that in the csv, if so I printed out the plaintext phone number and vote. I got a total runtime of 44 seconds when I ran this script on the lab machines.