Tony Majestro

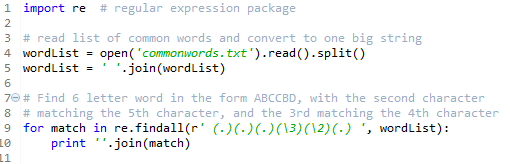
Problem Set 1

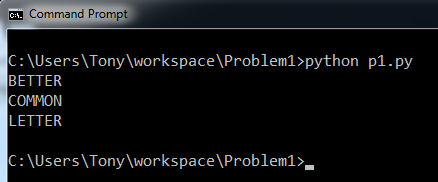
1. To begin decrypting the message, I tried to print out all possible plaintexts for a shift cipher with . This did not produce a readable message, so I assumed the message was encrypted with a substitution cipher.

I tried to take advantage of the fact that an encrypted character will always be the same throughout the encrypted message, so I could search through a dictionary for words that had similar formats to certain English words.

The first word is **BPKKPM**, and the fifth word is **YMBPKKPM**, which means that the fifth word is the same as the first word with a two letter prefix. Searching through all of the words in a dictionary of 1000 common English words

(<http://www.giwersworld.org/computers/linux/common-words.phtml>) for this particular pattern gives the following matches:

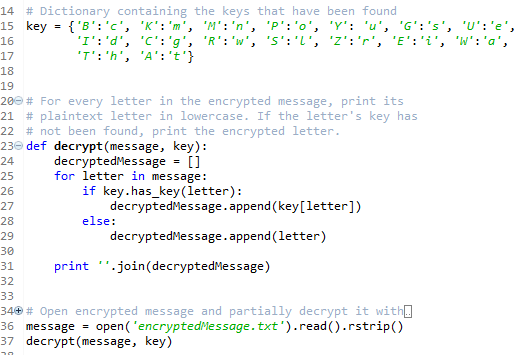


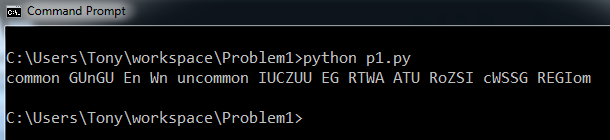
0

The first word is probablyCOMMON, because it is the only word that could have a prefix (*UNCOMMON*). So the first word is COMMON and the fifth word is UNCOMMON. This gives us the following key:

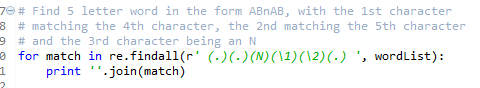
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B |  |  |  |  |  |  |  |  |  | K | M | P |  |  |  |  |  | Y |  |  |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |

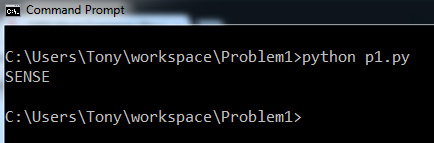
After partially decrypting the message, we are left with:





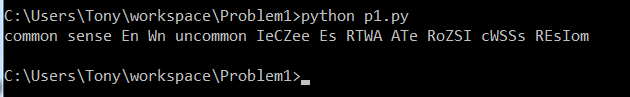
Next, we can attempt to find the second word, because it only has two missing characters in a very specific format. Searching for the possible words in this format gives only one match: SENSE.





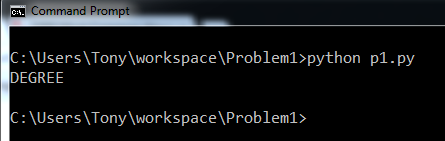
Now we have updated our key and have decrypted more of the message:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B |  | U |  |  |  |  |  |  |  | K | M | P |  |  | Z | G |  | Y |  |  |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |



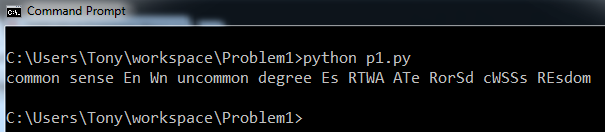
Searching for the sixth word gives us DEGREE:





After updating the key again, we have the partially decrypted message:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B | I | U |  | C |  |  |  |  |  | K | M | P |  |  | Z | G |  | Y |  |  |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |



RosSd looks easy enough to decrypt:

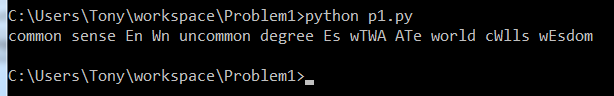


This gives the output:



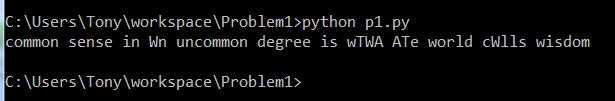
We can update our key again and partially decrypt the message:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B | I | U |  | C |  |  |  |  | S | K | M | P |  |  | Z | G |  | Y |  | R |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |

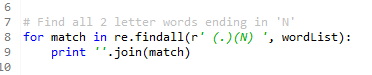


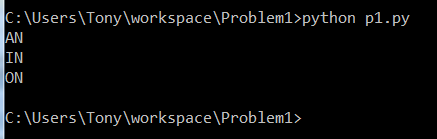
The last word is most likely WISDOM, so I’m not even going to run the script to verify it. We can update the key and partially decrypt again:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | B | I | U |  | C |  | E |  |  | S | K | M | P |  |  | Z | G |  | Y |  | R |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |



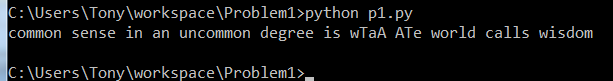
For the fourth word, there are only three 2 letter words ending in an ‘n’. It cannot be ‘in’ or ‘on’ because we have already found out that and . So the fourth word is AN and the second to last word is CALLS:





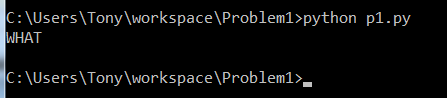
This gives us the new key and new decryption:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W |  | B | I | U |  | C |  | E |  |  | S | K | M | P |  |  | Z | G |  | Y |  | R |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |



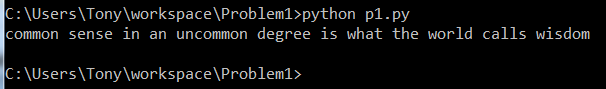
The only characters that have not been decrypted are ‘T’ and ‘A’. The eight word is probably WHAT and the ninth word is probably THE. A quick search proves this:





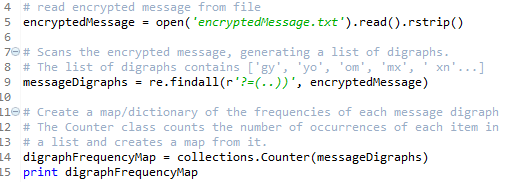
So we have the final key and the final decrypted message:

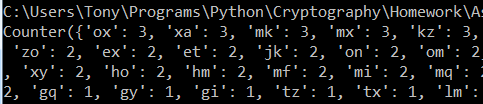
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W |  | B | I | U |  | C | T | E |  |  | S | K | M | P |  |  | Z | G | A | Y |  | R |  |  |  |
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |



1. The second problem involved decrypting a message that was encrypted with an affine cipher. The approach I took was to find the most common digraphs in the encrypted message and compare them to the most common digraphs in the English language.

I used a python script to find the most common digraphs in the encrypted message:



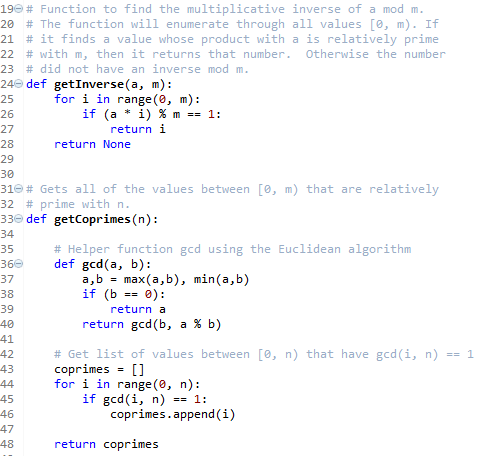


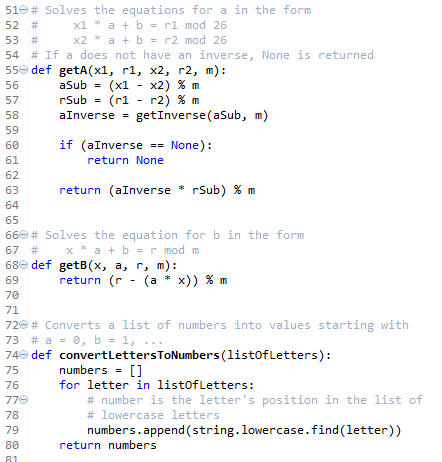
So OX, XA, MK, MX, KZ, … were the most frequently occurring digraphs in the message. Now that I know this, I can start with common digraphs in the encrypted message and use common digraphs in the English language to generate values for and for the Affine cipher. I can begin by guessing that O encrypts to T and X encrypts to H, since TH is the most frequently used digraph in the English language. This gives the equations:

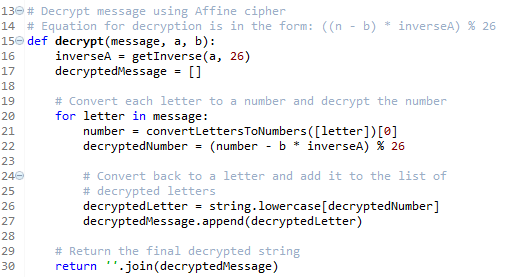
Solving for a gives us the equations:

However, the multiplicative inverse of 12 mod 26 does not exist because 12 and 26 are not relatively prime. Therefore, O cannot encrypt to T while X encrypts to H. But we can continue trying pairs of digraphs until we reach a valid value for . Once we’ve gotten a valid value for , we can solve for our and then decrypt the message. If the decrypted message is readable, then we know that we have valid values for and .

If the message is not readable, then there were incorrect values for a and b. I wrote a script to automate this process of finding the values of a and b that decrypt to the correct text. I have several helper functions for doing the algebra:

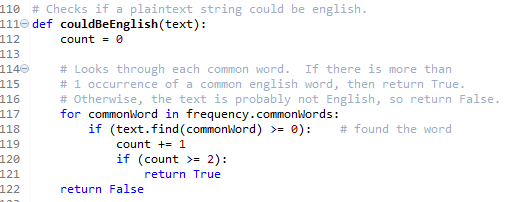


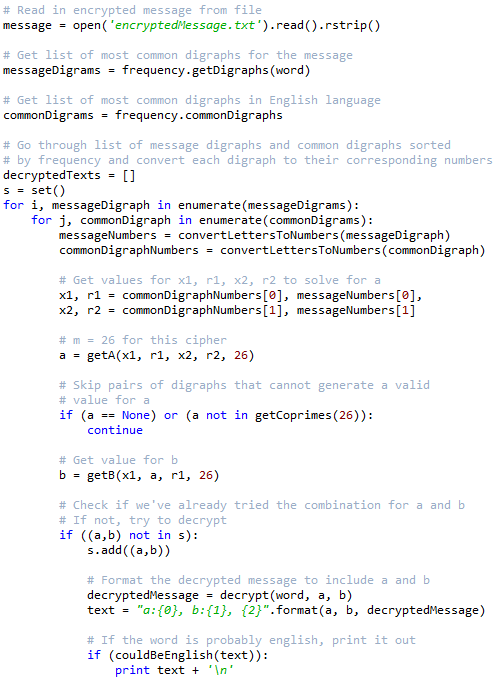




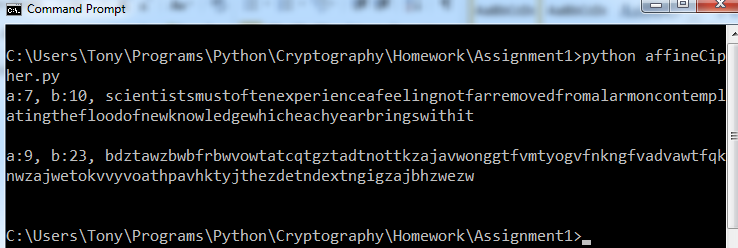
The main program goes through each pair of digraphs, searching for a valid pair of a and b. If the decryption looks like English, it gets printed out:







Only two of the pairs of a and b gave possible English values. The first is clearly English, decrypting to “Scientists must often experience a feeling not far removed from alarm on contemplating the flood of new knowledge which each year brings with it”, with values a = 7 and b = 10



1. The formula for the number of keys for an affine cipher is equal to the number of possible values for times the number of possible values for b.

The number of values for given is equal to the euler-phi function of m. That is, the number of values between [0, m) that are relatively prime to . The number of values for b given m is equal to , since the shift may be any value in [0, m). Therefore, the number of total keys is equal to the equation:

* Total keys = 30 \* 8 = 240
* Total keys = 100 \* 40 = 4000
* Total keys = 1225 \* 840 = 1,029,000

1. gcd(5847, 1410) =

gcd(1410, 5847 % 1410) =

gcd(1410, 207) =

gcd(207, 1410 % 207) =

gcd(168, 207 % 168) =

gcd(168, 39) =

gcd(39, 168 % 39) =

gcd(39, 12) =

gcd(12, 3) =

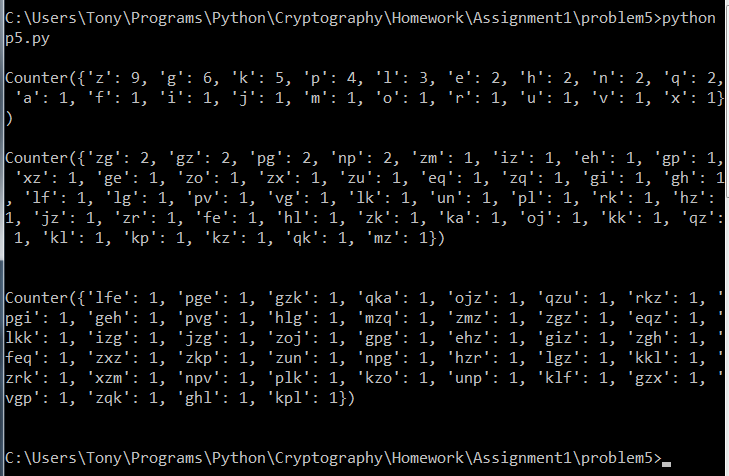
gcd(3, 12 % 3) =

gcd(3, 0)

gcd(5847, 1410) = 3

1. Because I did not know where words began or ended, I could not approach this problem like the first problem. Instead, I began this problem by looking at frequencies of letters, digraphs, and monographs:

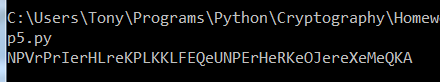




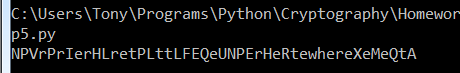
From here, I noticed several things.

* Z was by far the most popular letter, followed by G, Z, and P
* No digraph appeared more than twice
* GZ and ZG both appear twice, and look exactly like the two of the most frequent digraphs in the English language, ER and RE
* No trigraph appeared more than one time

I began by trying the digraph GZ and ZG. It appears that it could be RE and ER.



I noticed OJere, which would be some form of where. After attempting to try that key, I get the following partial decryption:



The work that I hand in is my own.

-Anthony Majestro