|  |  |
| --- | --- |
| Report | |
| File Encryption And Disk Encryption For Maximum Security | |
|  |  |

## Task 1

The Brute Force attack on Caesar Cipher was used to decrypt the ciphertext.

“ te td fyvyzhy szh pqqpnetgp esp nlpdlc ntaspc hld le esp etxp, mfe te td wtvpwj

ez slgp mppy cpldzylmwj dpnfcp, yze wplde mpnlfdp xzde zq nlpdlc'd pypxtpd

hzfwo slgp mppy twwtepclep lyo zespcd hzfwo slgp lddfxpo esle esp xpddlrpd

hpcp hcteepy ty ly fyvyzhy qzcptry wlyrflrp. “

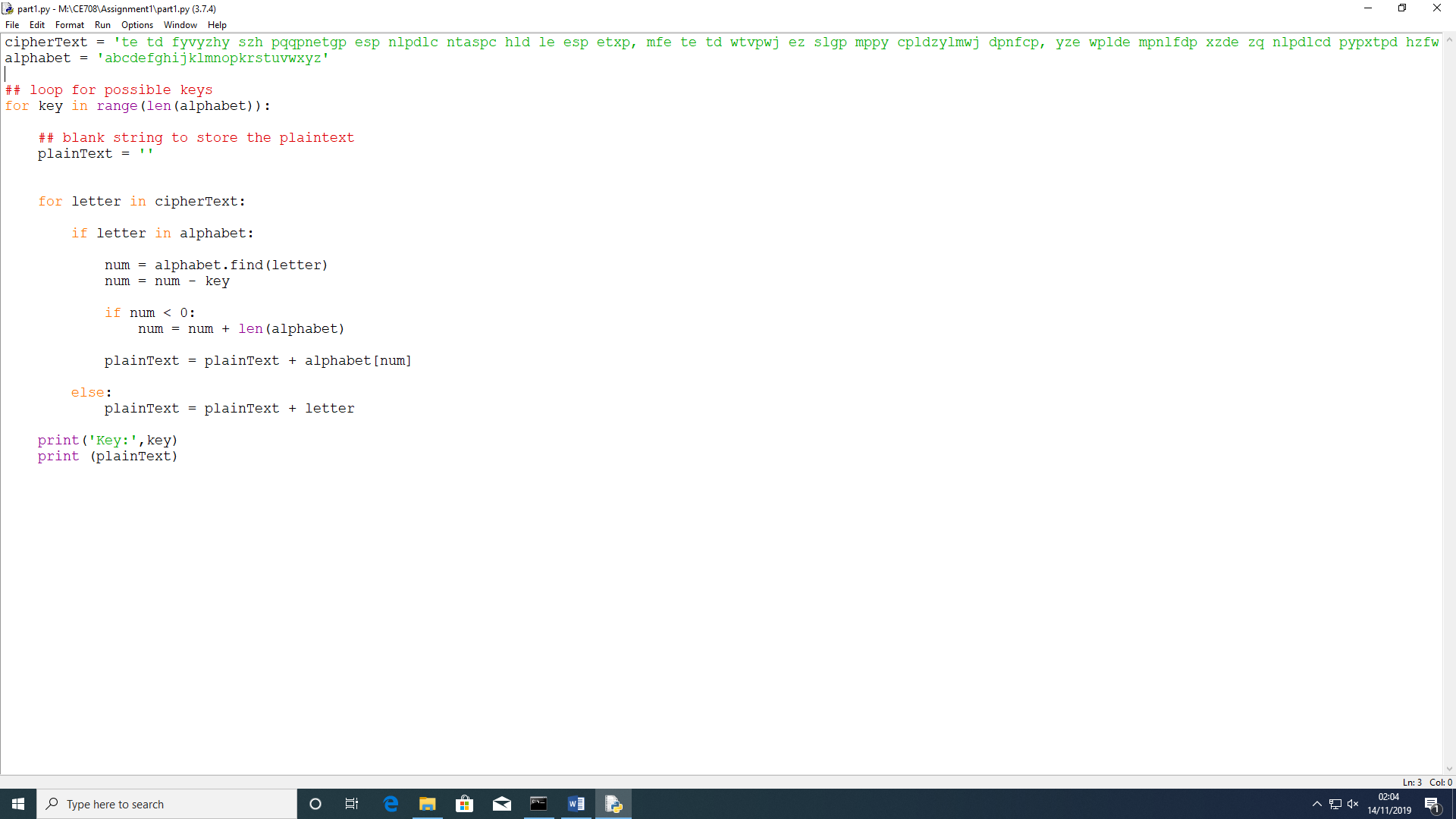
The result plaintext was from key 11:

“it is unknown how eqqective the caesar cipher was at the time, but it is likely to have been reasonably secure, not least because most oq caesars enemies would have been illiterate and others would have assumed that the messages were written in an unknown qoreign language. “

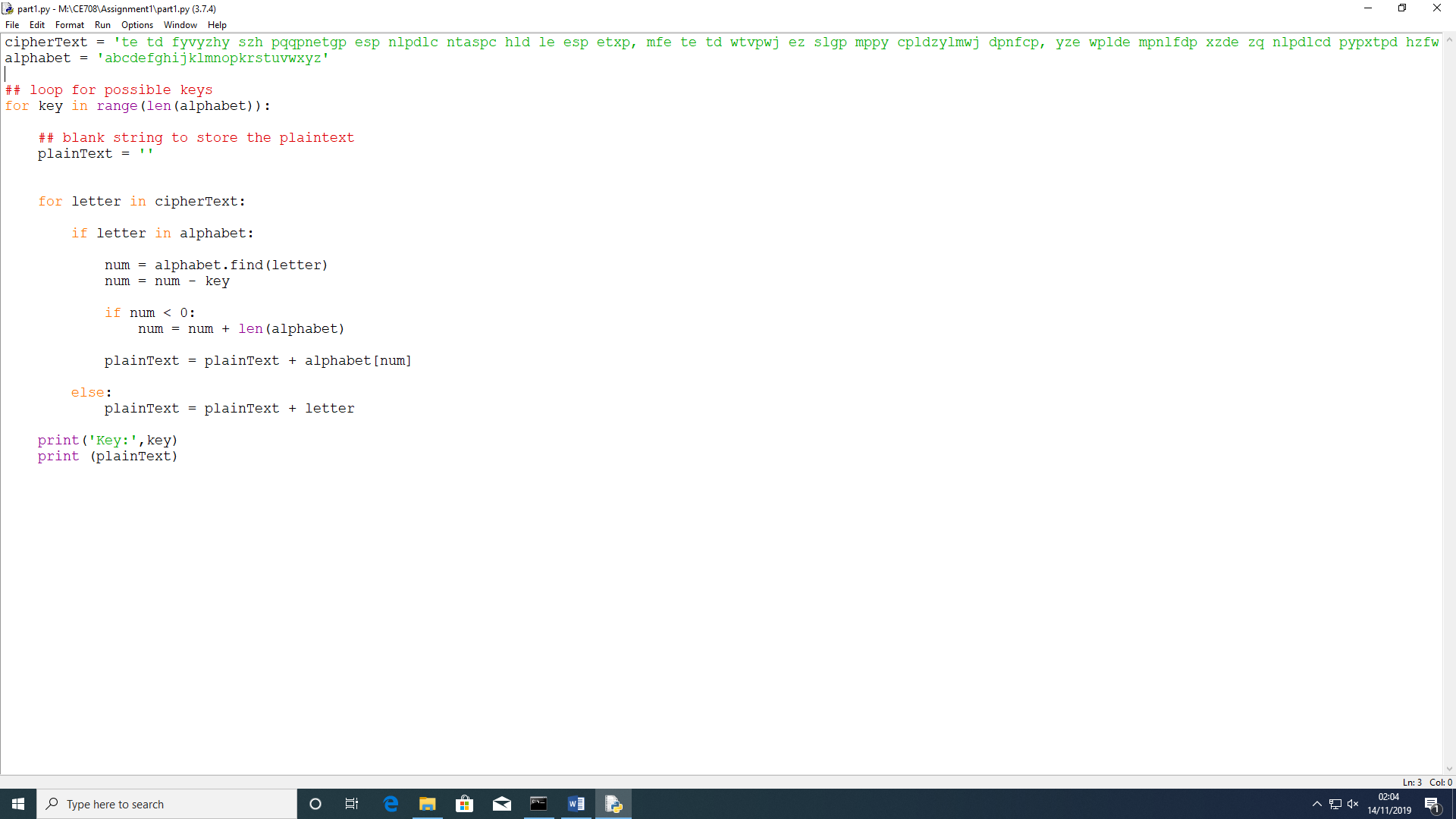
The reason brute force attack was used is because the key was not given. As a result, every possible decryption key was used to decrypt the ciphertext by looking at the output of each key until the plaintext made some sense.

The program creates a variable called “cipherText” to store the ciphertext string that needs to be decrypted. Then another variable called “alphabet” is created to store every letter that can be encrypted. Then a for loop is created to iterate over the return value from a call to a function named range(). So range values can be used to loop a specific number of times. In this case up to 26 times. This was not hardcoded to the value 26 because if someone was to modify the variable ‘alphabet’, then it would not work. The first time the key will be set to 0 and the ciphertext will be decrypted with the key 0. The next iteration will be with key 1 and so on.

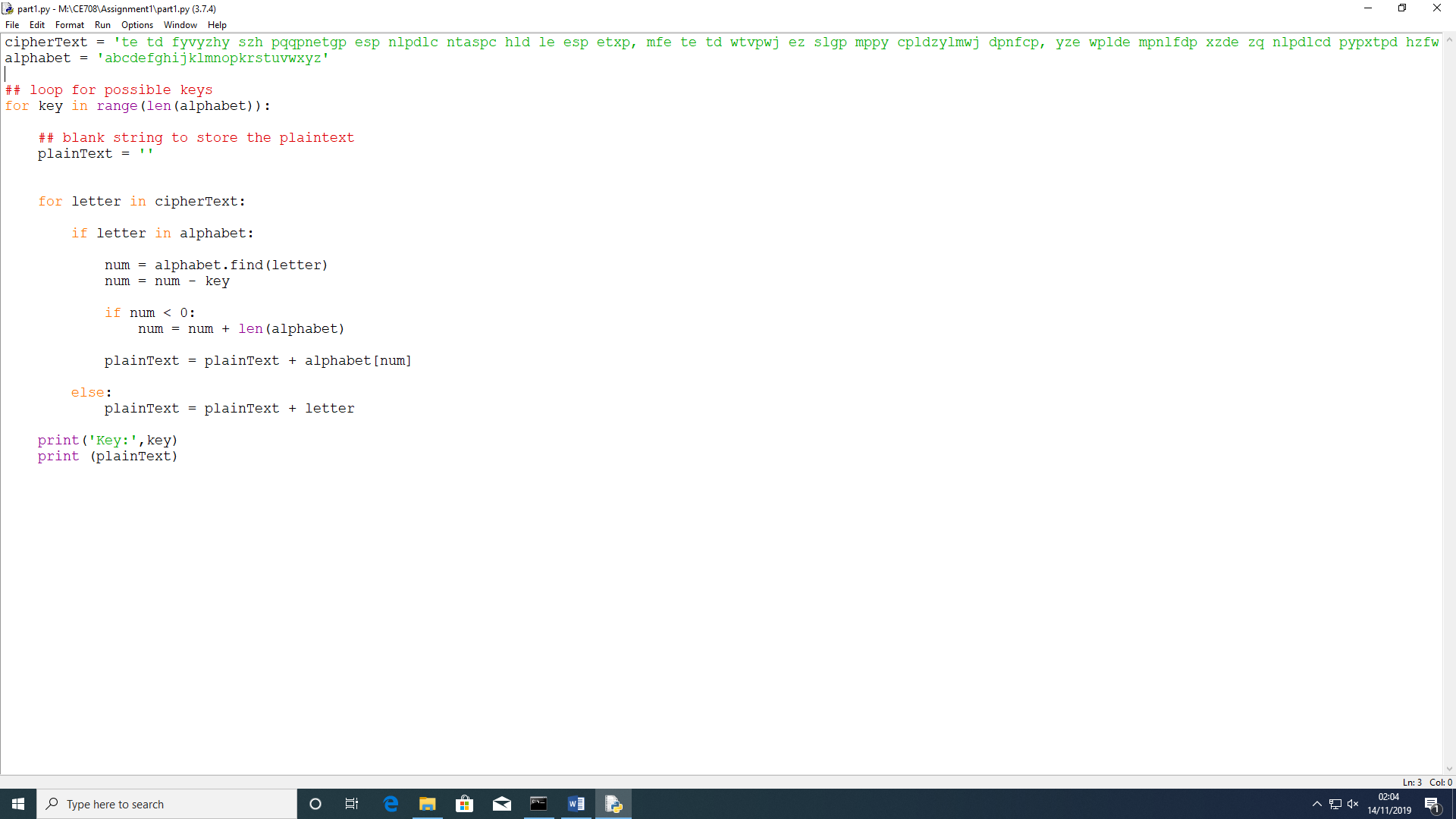
The variable “plainText” is a blank string used to store the plaintext. The next loop goes through every letter in the ciphertext. On each iteration, it checks if the letter is lowercase and also if it is in the ‘alphabet’ variable. If it is then it will decrypt it. If not then the “else” statement at the bottom won’t carry on with the decryption process and just adds letters to the end of plaintext.

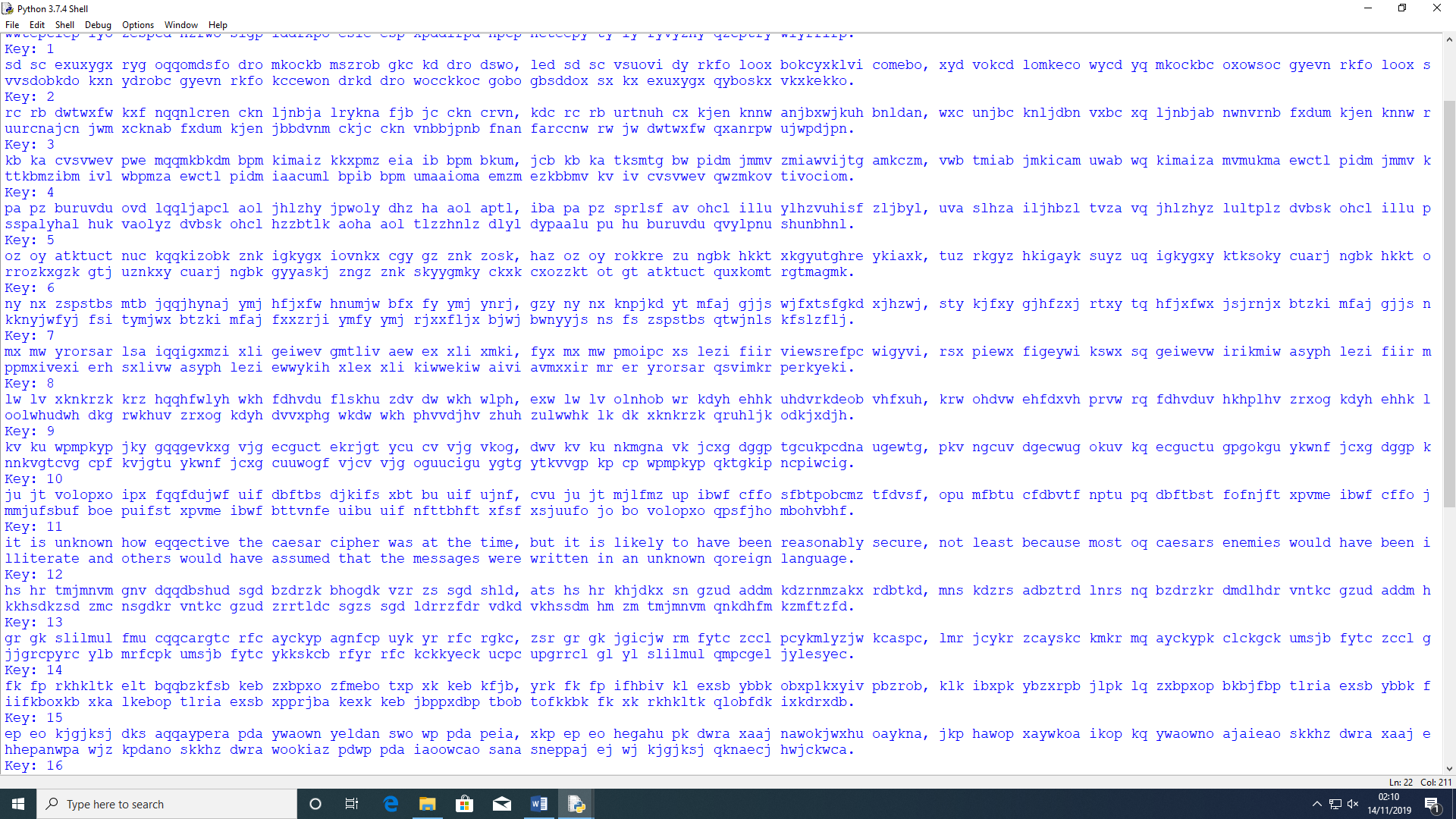


The next line will locate where letter is in the alphabet with the find() method and store it in a variable called number. The key is then subtracted from ‘number’. Subtracting the key will decrypt and adding the key will encrypt in Caesar Cipher. However, variable number may end up being a negative number. So the ‘if number < 0’ will check this and add 26 if it was less than 0.



The next line adds the decrypted message to the end of the blank string called “plainText”. Then the key is printed with the plaintext under it.

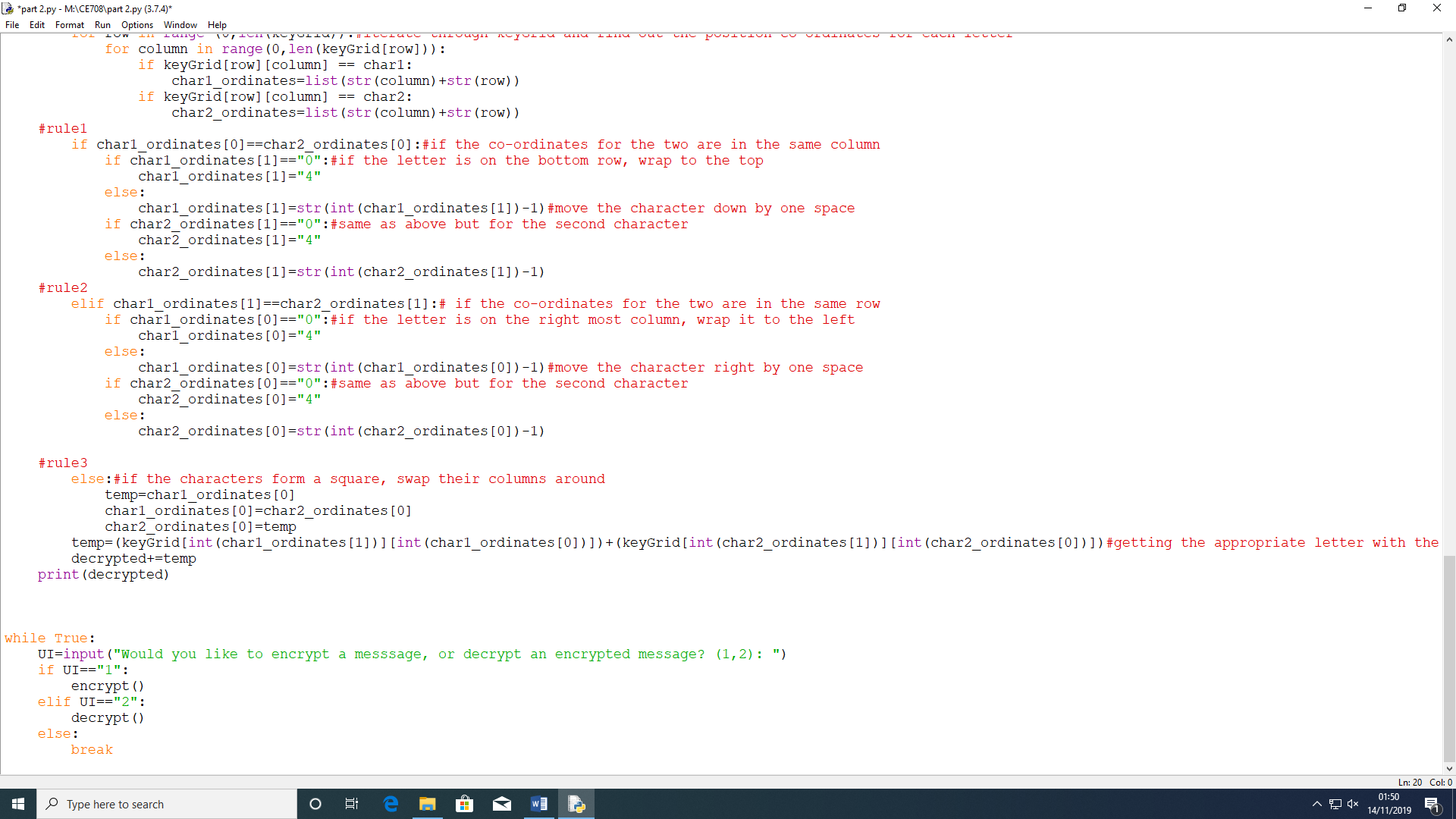


Output:

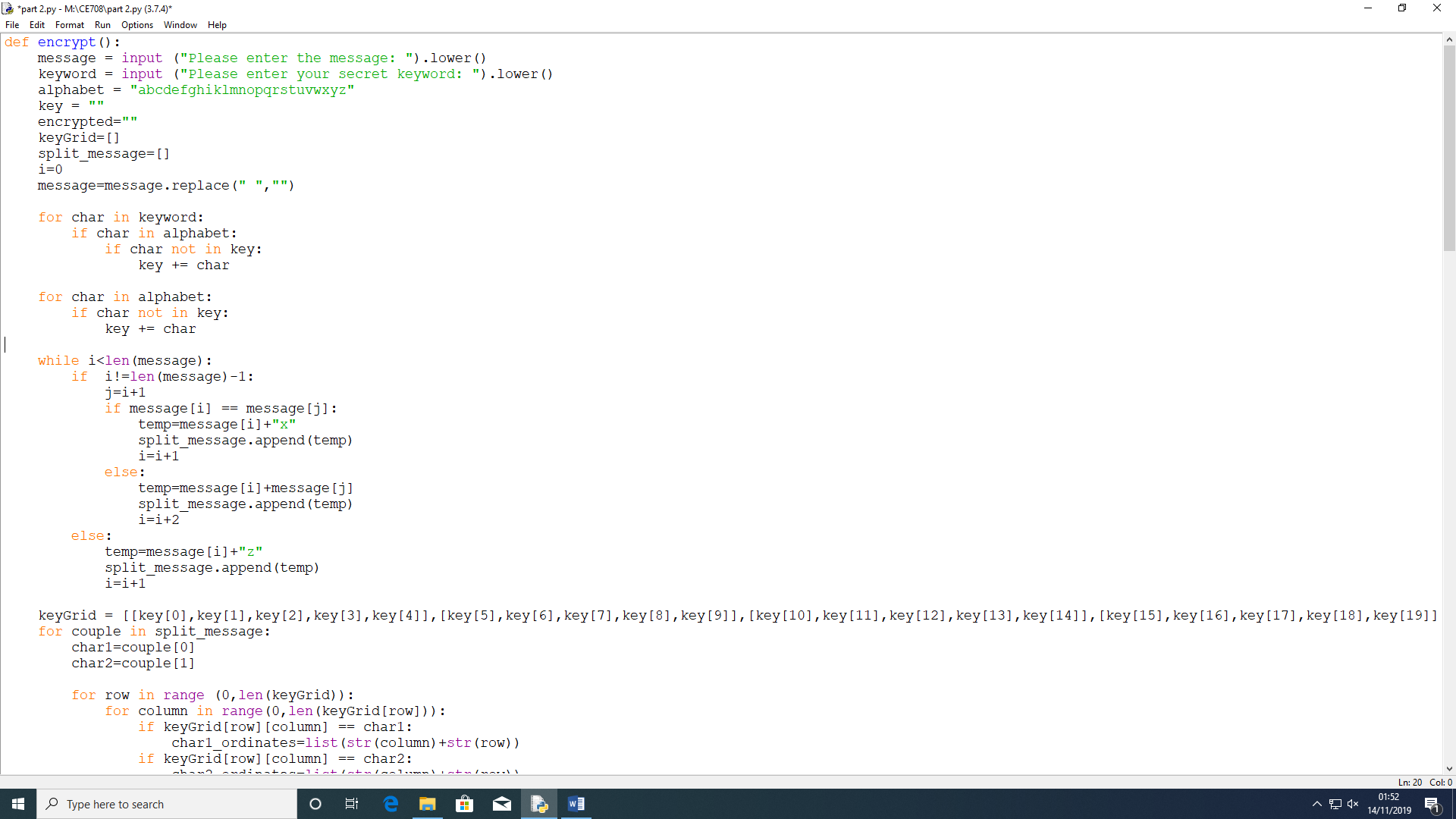
## Task 2

For part 2 of the assignment, Playfair Cipher was used to encrypt a plaintext that the user enters with the key that the user also enters. Also, using Playfair Cipher a plaintext was decrypted.

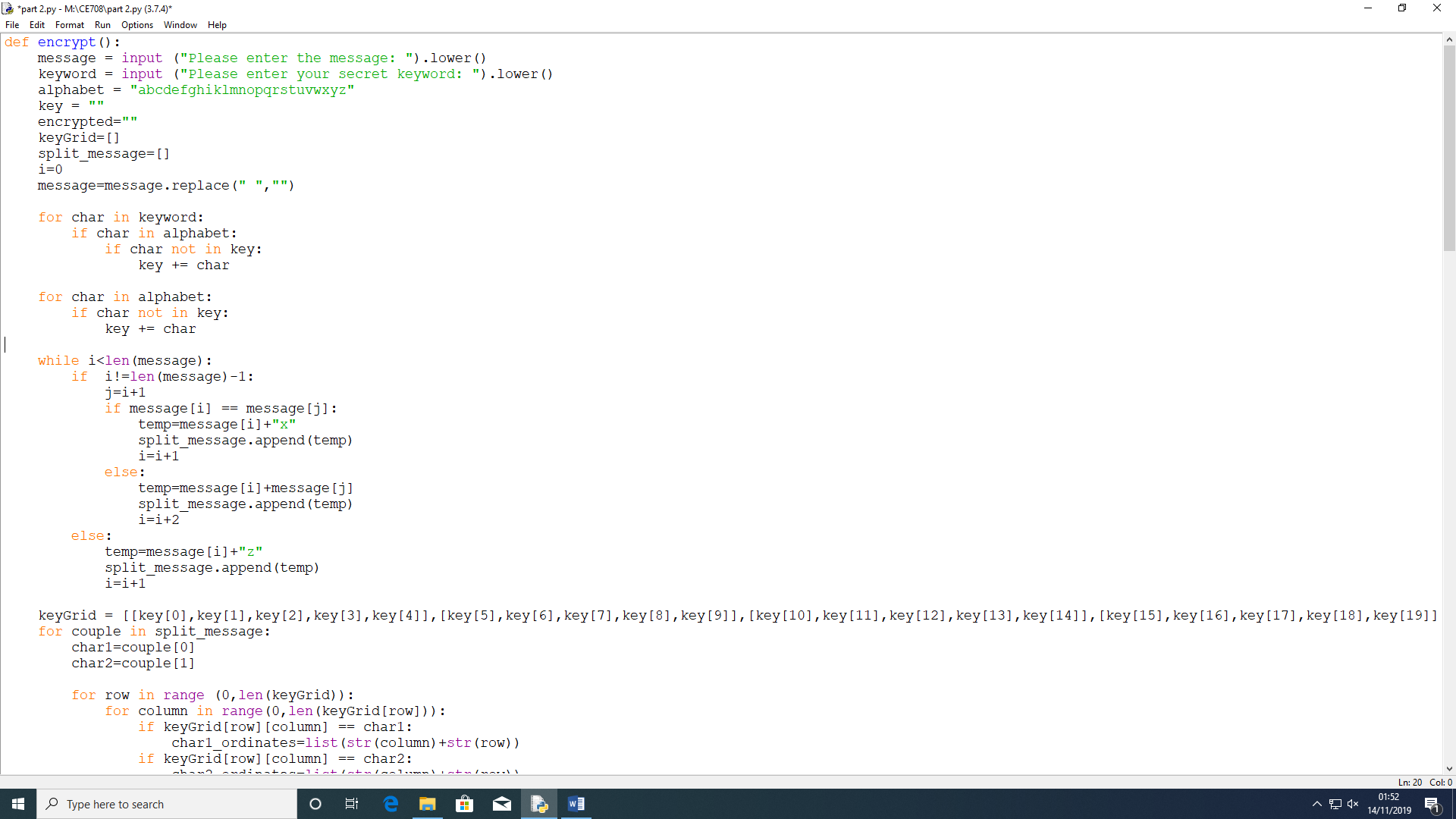
The program was split into 2 functions – encrypt() and decrypt() and the user can choose which function to invoke through a user interface that was created. The user is asked whether they would like to encrypt or decrypt by entering 1 or 2.



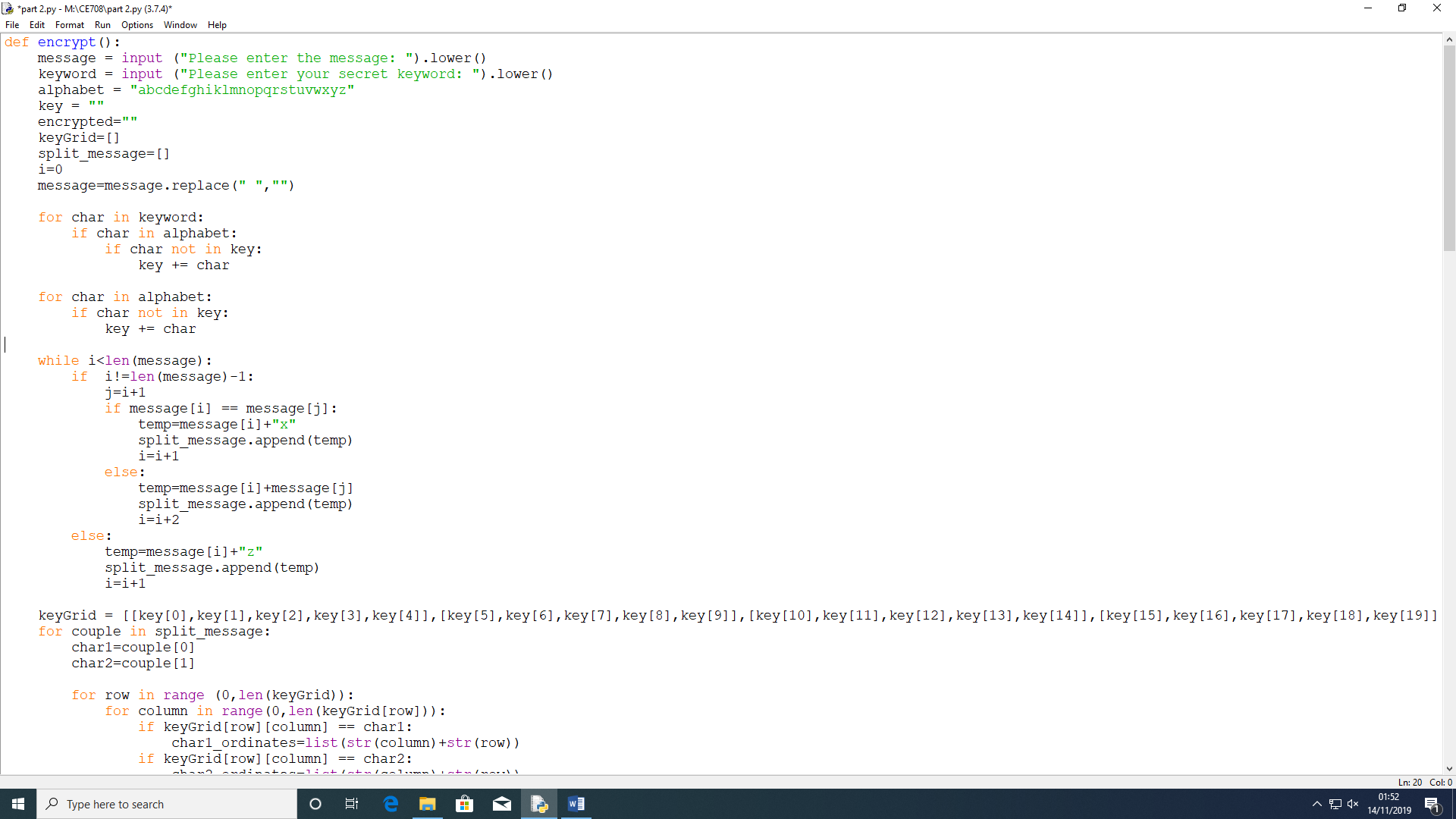
If the user chooses to encrypt then the program will allow them to enter the message to encrypt. Then they would be asked to enter the keyword to encrypt with. Both are changed to lower case. Then the following variable are defined – ‘alphabet’ stores every letter of the alphabet expect ‘j’. An empty string ‘key’ which will consist of a combination of keyword alphabet to make a key. An empty string ‘encrypted’ which will be populated with the encrypted message. An empty list ‘keyGrid’ that will consist of lists to make a key grid. An empty list that will split the message per 2 letters and insert an x or z where necessary. An iterator ‘i’. Then ‘messages’ which removes white spaces.



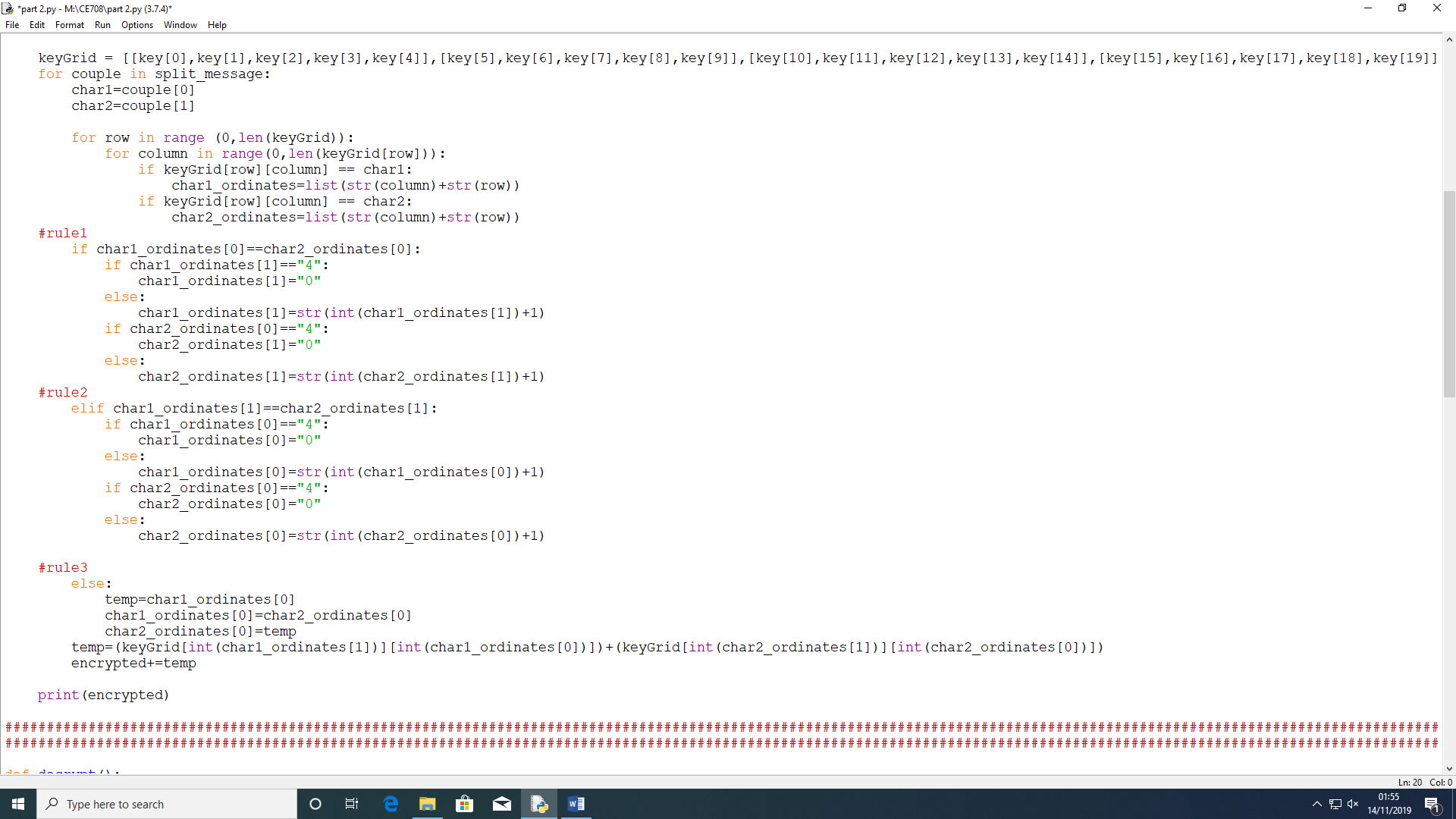
The for loops are to make the key to encrypt the message. So for every letter in keyword, if the letter is in the alphabet but not in the ‘key’ variable then adds the letter to the key variable. The next for loops checks every letter in the alphabet and if the letter is not in the key then adds the letter to the key. E.g. if the keyword is ‘keyword’ then the key will now be ‘keywordabcfghilmnpqstuvxz’.



The while statement is to check if the counter is in range of the length of the message. If the pointer is not pointing to the last letter then the variable is set to point to the letter to the right. If the two letters are the same, x is added at the end of the first letter and the iterator is increased by 1. If not, then adds the two letters to the list and increases the iterator by 2. If it is the last letter and it is not pairs, ‘z’ is added at the end of it. The iterator is increased by 1 so it loops out of the while loop.



The keyGrid variable consists of a list of lists with the new key, making the matrix. Then, for loop is created to go through every pair in the split message (variable of empty list that will split the message per 2 letters and insert an x or z where necessary). A for loop is created again to iterate through ‘keyGrid’ and find out the position co-ordinates for each letter.



The program then proceeds with the three rules of Playfair Cipher.

The first rule:

If the co-ordinates for the two are in the same column, each letter must moves down one. If the letter is on the bottom row then it wraps to the top. Then an else statement is created to move the character/letter up by one space and the same is done for the second character.

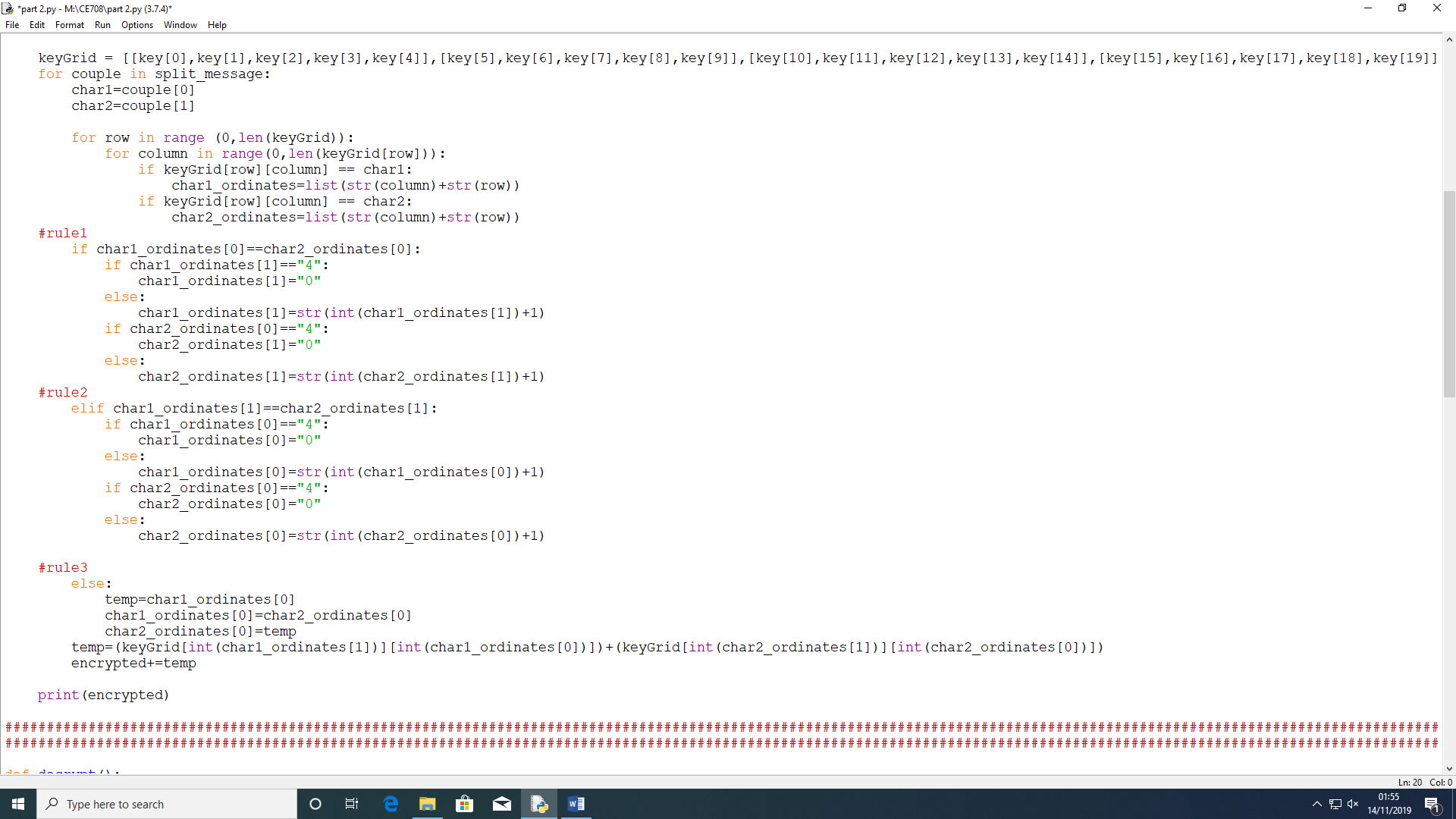
Second Rule:

Else if the co-ordinates for the two are in the same row, each letter moves one to the right. If the letter is on the right most column then it wraps it to the left. Then the else statement moves the character right by one space. The same is done for the second character.

Third Rule

Else if it forms a rectangle then the letters are swapped with the ones on the end of the rectangle. This is done by swapping their columns.

The variable ‘temp’ is getting the appropriate letter with the co-ordinates calculated. Then all this is added to the variable ‘encrypted’ which is then printed to the screen.



The decryption function is the exact same as the encryption function but the rules are different and opposite.

The first rule:

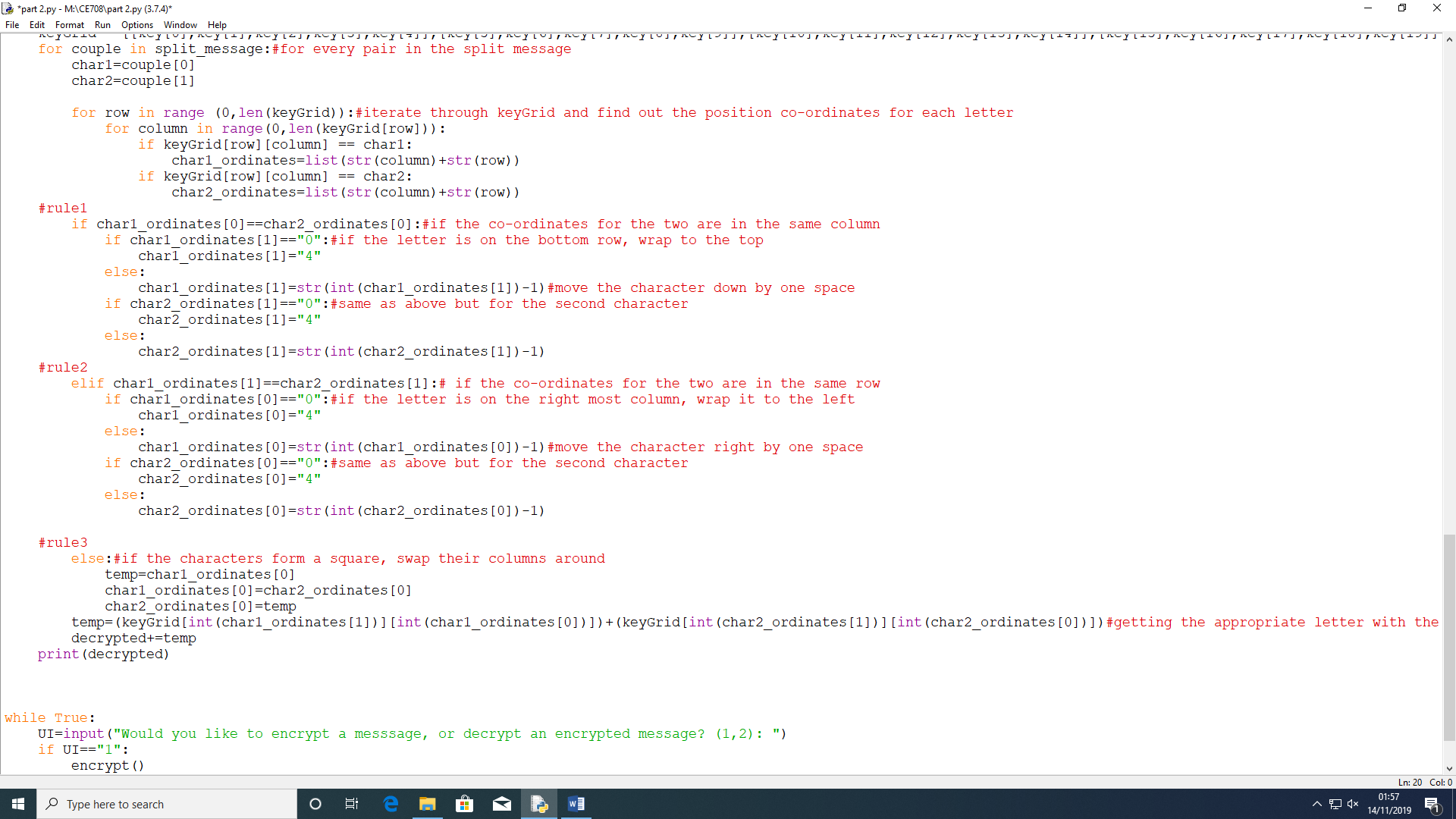
If the co-ordinates for the two are in the same column, each letter must moves up one. If the letter is on the top row then it wraps to the bottom. Then an else statement is created to move the character/letter up by one space and the same is done for the second character.

Second Rule:

Else if the co-ordinates for the two are in the same row, each letter moves one to the left. If the letter is on the left most column then it wraps it to the right. Then the else statement moves the character left by one space. The same is done for the second character.

Third Rule

Else if it forms a rectangle then the letters are swapped with the ones on the end of the rectangle. This is done by swapping their columns.



Output:

