codecademy

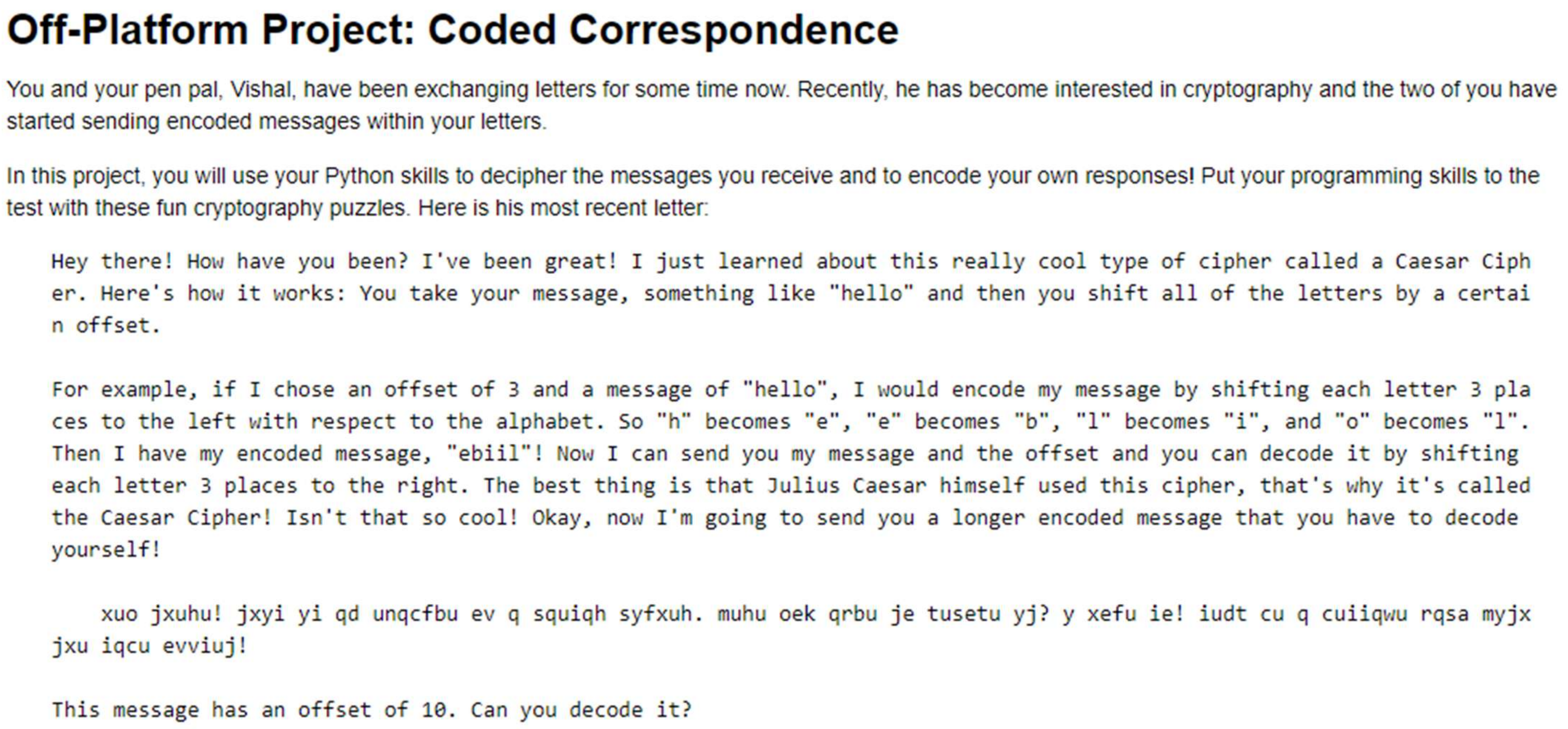
Coded Correspondence

Python project

Knox Maclean Bather

09/10/2023

Project Explaination



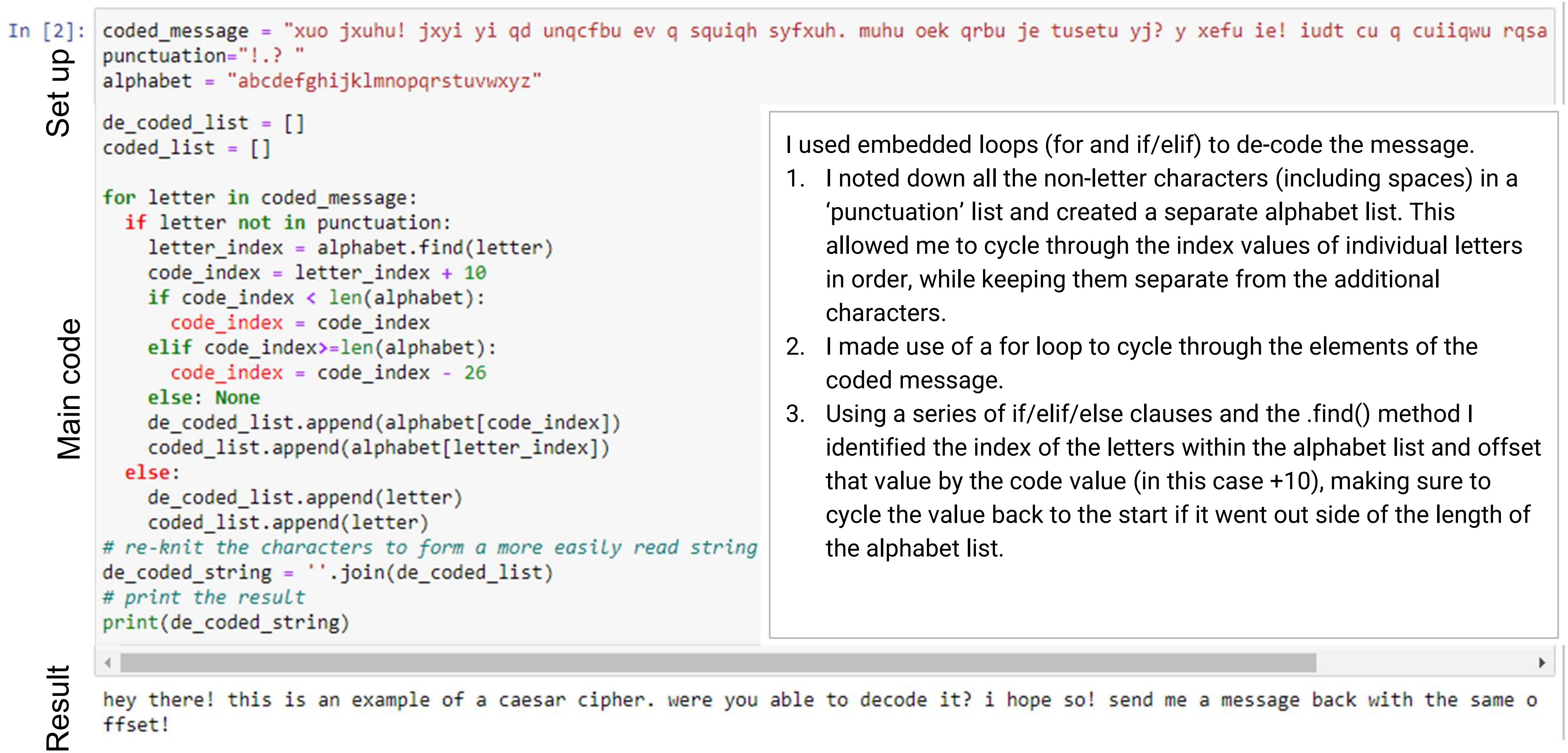
Off-Platform Project: Coded Correspondence

You and your pen pal, Vishal, have been exchanging letters for some time now. Recently, he has become interested in cryptography and the two of you have started sending encoded messages within your letters.

In this project, you will use your Python skills to decipher the messages you receive and to encode your own responses! Put your programming skills to the test with these fun cryptography puzzles. Here is his most recent letter:

```
Hey there! How have you been? I've been great! I just learned about this really cool type of cipher called a Caesar Ciph
er. Here's how it works: You take your message, something like "hello" and then you shift all of the letters by a certai
n offset.

For example, if I chose an offset of 3 and a message of "hello", I would encode my message by shifting each letter 3 pla
ces to the left with respect to the alphabet. So "h" becomes "e", "e" becomes "b", "l" becomes "i", and "o" becomes "l".
Then I have my encoded message, "ebiil"! Now I can send you my message and the offset and you can decode it by shifting
each letter 3 places to the right. The best thing is that Julius Caesar himself used this cipher, that's why it's called
the Caesar Cipher! Isn't that so cool! Okay, now I'm going to send you a longer encoded message that you have to decode
yourself!

    xuo jxuhu! jxyi yi qd unqcfbu ev q squiqh syfxuh. muhu oek qrbu je tusetu yj? y xefu ie! iudt cu q cuiiqwu rqsa myjx
jxu iqcu evviuj!

This message has an offset of 10. Can you decode it?
```

1. De-code Vishal’s message.



Set up

Main code

Result

```
In [2]: coded_message = "xuo jxuhu! jxyi yi qd unqcfbu ev q squiqh syfxuh. muhu oek qrbu je tusetu yj? y xefu ie! iudt cu q cuiiqwu rqsa
        punctuation="!.? "
        alphabet = "abcdefghijklmnopqrstuvwxyz"

        de_coded_list = []
        coded_list = []

        for letter in coded_message:
          if letter not in punctuation:
            letter_index = alphabet.find(letter)
            code_index = letter_index + 10
            if code_index < len(alphabet):
              code_index = code_index
            elif code_index>=len(alphabet):
              code_index = code_index - 26
            else: None
            de_coded_list.append(alphabet[code_index])
            coded_list.append(alphabet[letter_index])
          else:
            de_coded_list.append(letter)
            coded_list.append(letter)
        # re-knit the characters to form a more easily read string
        de_coded_string = ''.join(de_coded_list)
        # print the result
        print(de_coded_string)
```

```
hey there! this is an example of a caesar cipher. were you able to decode it? i hope so! send me a message back with the same o
ffset!
```

I used embedded loops (for and if/elif) to de-code the message.

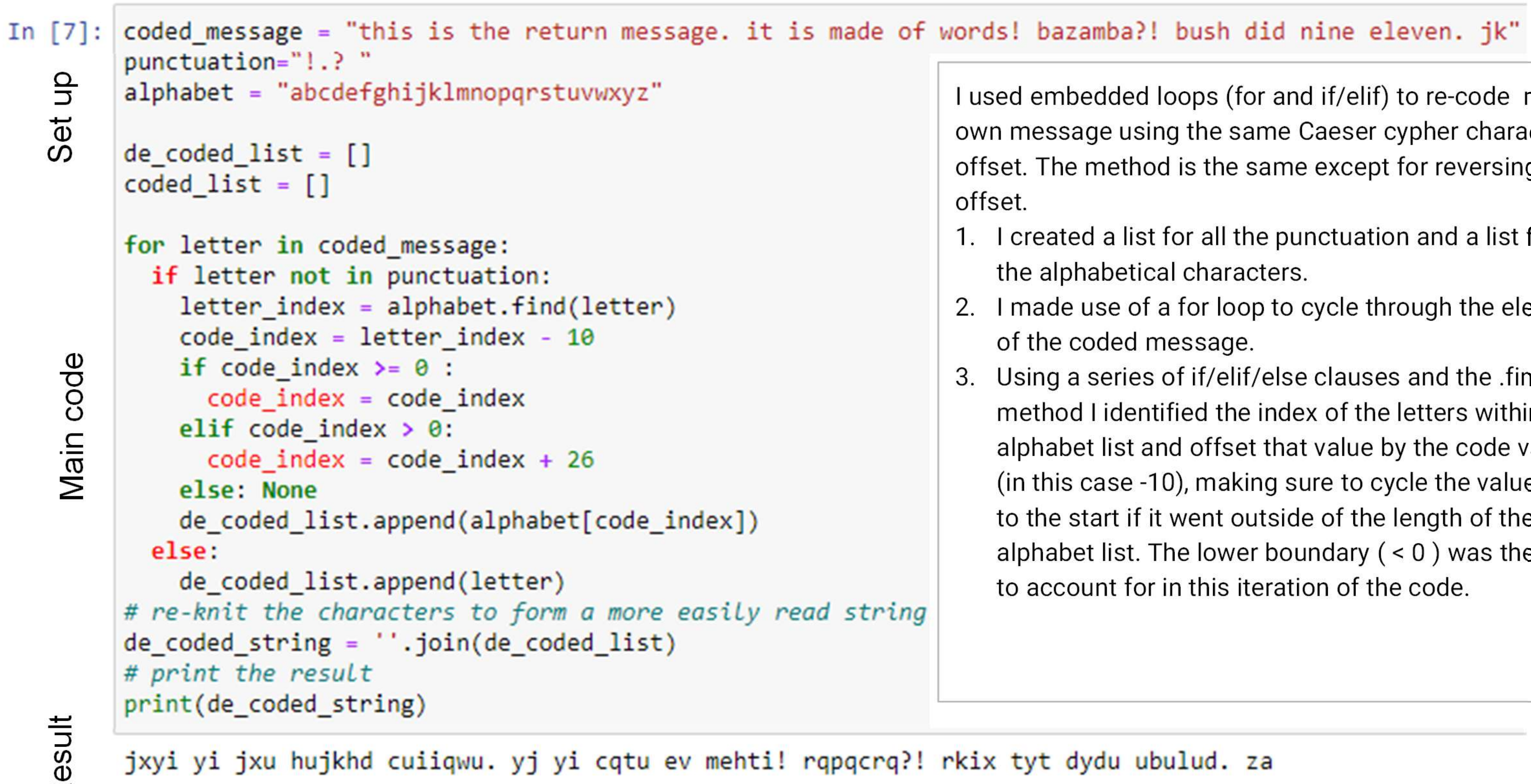
1. I noted down all the non-letter characters (including spaces) in a ‘punctuation’ list and created a separate alphabet list. This allowed me to cycle through the index values of individual letters in order, while keeping them separate from the additional characters.
2. I made use of a for loop to cycle through the elements of the coded message.
3. Using a series of if/elif/else clauses and the .find() method I identified the index of the letters within the alphabet list and offset that value by the code value (in this case +10), making sure to cycle the value back to the start if it went out side of the length of the alphabet list.

2. Encode your own message to send back to Vishal.

Set up

Main code

Result



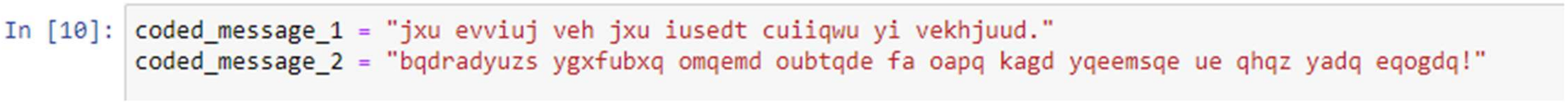
```
In [7]: coded_message = "this is the return message. it is made of words! bazamba?! bush did nine eleven. jk"
punctuation="!.? "
alphabet = "abcdefghijklmnopqrstuvwxyz"

de_coded_list = []
coded_list = []

for letter in coded_message:
  if letter not in punctuation:
    letter_index = alphabet.find(letter)
    code_index = letter_index - 10
    if code_index >= 0 :
      code_index = code_index
    elif code_index > 0:
      code_index = code_index + 26
    else: None
    de_coded_list.append(alphabet[code_index])
  else:
    de_coded_list.append(letter)
# re-knit the characters to form a more easily read string
de_coded_string = ''.join(de_coded_list)
# print the result
print(de_coded_string)
```

```
jxyi yi jxu hujkhd cuiiqwu. yj yi cqtu ev mehti! rqpqcrq?! rkix tyt dydu ubulud. za
```

I used embedded loops (for and if/elif) to re-code my own message using the same Caeser cypher character offset. The method is the same except for reversing the offset.

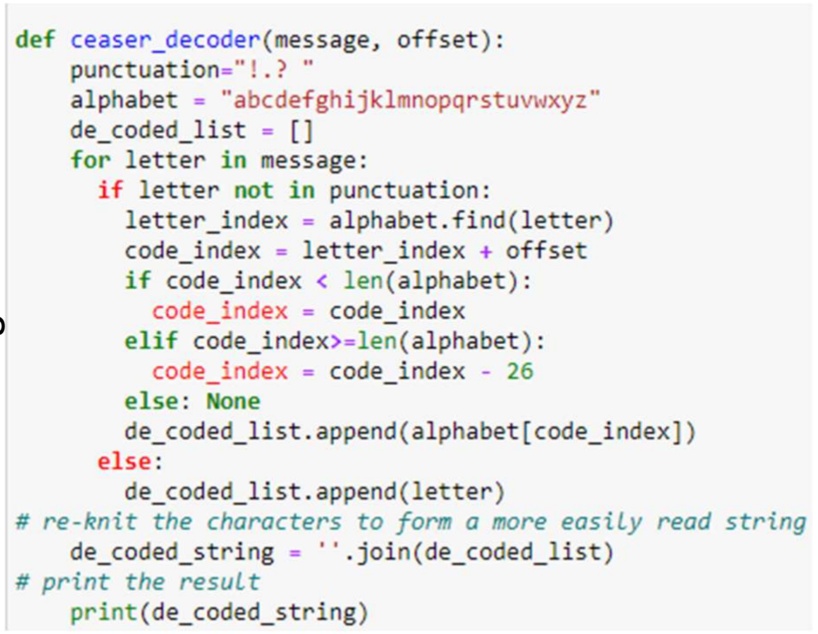
1. I created a list for all the punctuation and a list for all the alphabetical characters.
2. I made use of a for loop to cycle through the elements of the coded message.
3. Using a series of if/elif/else clauses and the .find() method I identified the index of the letters within the alphabet list and offset that value by the code value (in this case -10), making sure to cycle the value back to the start if it went outside of the length of the alphabet list. The lower boundary (< 0) was the one to account for in this iteration of the code.

3. De-code two messages and create a function for coding and decoding messages.



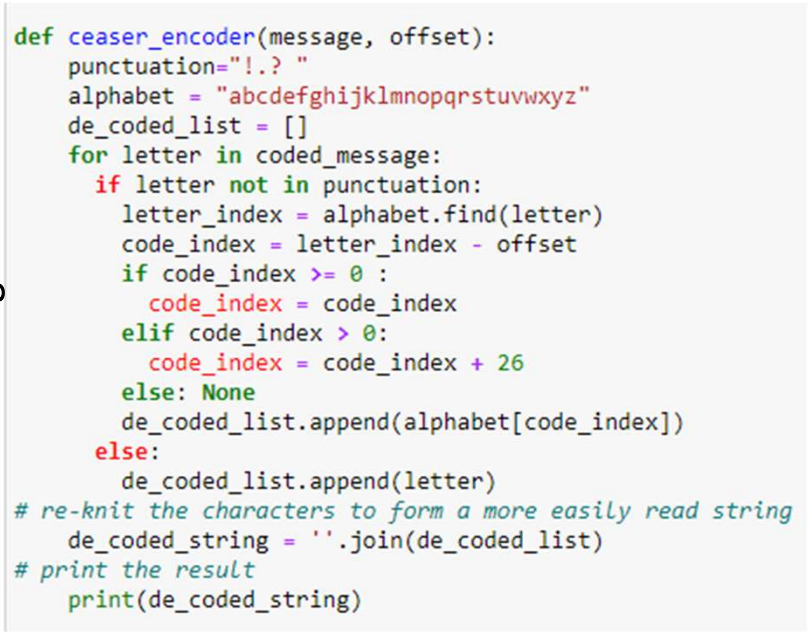
```
In [10]: coded_message_1 = "jxu evviuj veh jxu iusedt cuiiqwu yi vekhjuud."
         coded_message_2 = "bqdradyuzs ygxfubxq omqemd oubtqde fa oapq kagd yqeemsqe ue qhqz yadq eqogdq!"
```

De-coding function

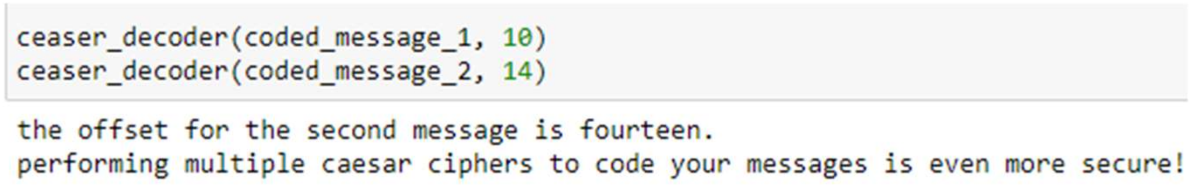


```
def ceaser_decoder(message, offset):
    punctuation="!.? "
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    de_coded_list = []
    for letter in message:
      if letter not in punctuation:
        letter_index = alphabet.find(letter)
        code_index = letter_index + offset
        if code_index < len(alphabet):
          code_index = code_index
        elif code_index>=len(alphabet):
          code_index = code_index - 26
        else: None
        de_coded_list.append(alphabet[code_index])
      else:
        de_coded_list.append(letter)
# re-knit the characters to form a more easily read string
    de_coded_string = ''.join(de_coded_list)
# print the result
    print(de_coded_string)
```

En-coding function



```
def ceaser_encoder(message, offset):
    punctuation="!.? "
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    de_coded_list = []
    for letter in coded_message:
      if letter not in punctuation:
        letter_index = alphabet.find(letter)
        code_index = letter_index - offset
        if code_index >= 0 :
          code_index = code_index
        elif code_index > 0:
          code_index = code_index + 26
        else: None
        de_coded_list.append(alphabet[code_index])
      else:
        de_coded_list.append(letter)
# re-knit the characters to form a more easily read string
    de_coded_string = ''.join(de_coded_list)
# print the result
    print(de_coded_string)
```

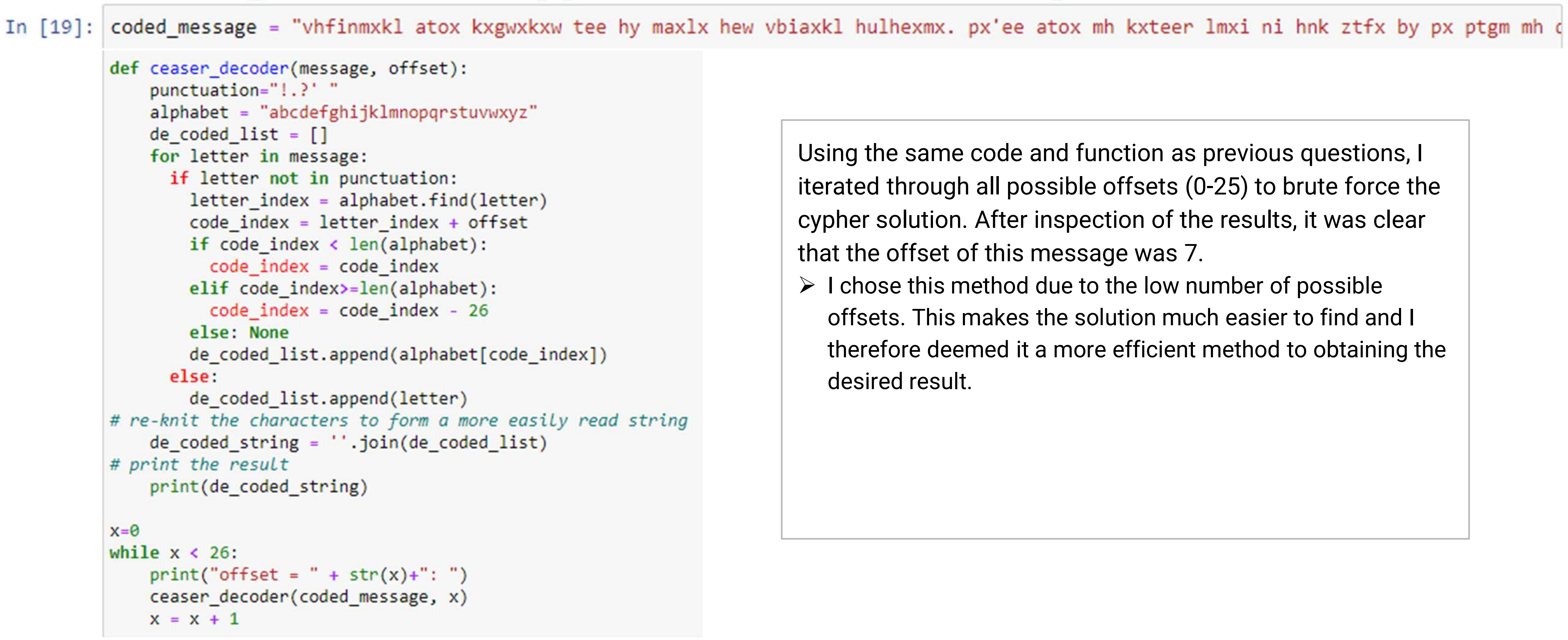


```
ceaser_decoder(coded_message_1, 10)
ceaser_decoder(coded_message_2, 14)
```

```
the offset for the second message is fourteen.
performing multiple caesar ciphers to code your messages is even more secure!
```

Created two function for the de/en-coding of messages using the Caeser cypher. The variables of the functions were the string message and the offset of encryption.

4. Solving a Caeser cypher without knowing the offset.

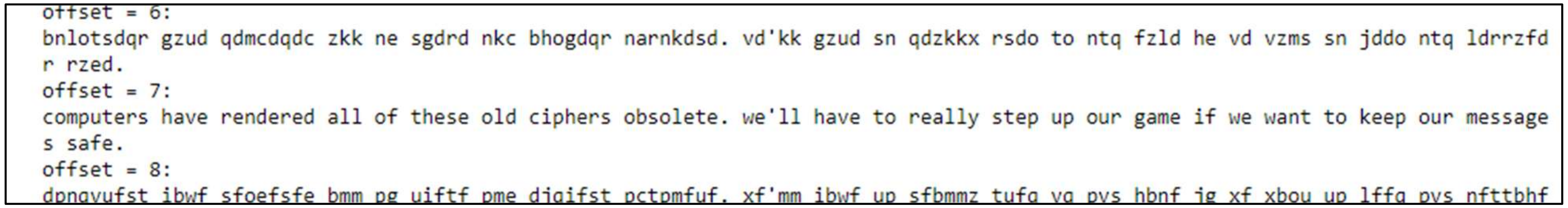


```
In [19]: coded_message = "vhfinmxkl atox kxgwxkxw tee hy maxlx hew vbiaxkl hulhexmx. px'ee atox mh kxteer lmxi ni hnk ztfx by px ptgm mh c

         def ceaser_decoder(message, offset):
             punctuation="!.?' "
             alphabet = "abcdefghijklmnopqrstuvwxyz"
             de_coded_list = []
             for letter in message:
               if letter not in punctuation:
                 letter_index = alphabet.find(letter)
                 code_index = letter_index + offset
                 if code_index < len(alphabet):
                   code_index = code_index
                 elif code_index>=len(alphabet):
                   code_index = code_index - 26
                 else: None
                 de_coded_list.append(alphabet[code_index])
               else:
                 de_coded_list.append(letter)
         # re-knit the characters to form a more easily read string
             de_coded_string = ''.join(de_coded_list)
         # print the result
             print(de_coded_string)

         x=0
         while x < 26:
             print("offset = " + str(x)+": ")
             ceaser_decoder(coded_message, x)
             x = x + 1
```

Using the same code and function as previous questions, I iterated through all possible offsets (0-25) to brute force the cypher solution. After inspection of the results, it was clear that the offset of this message was 7.

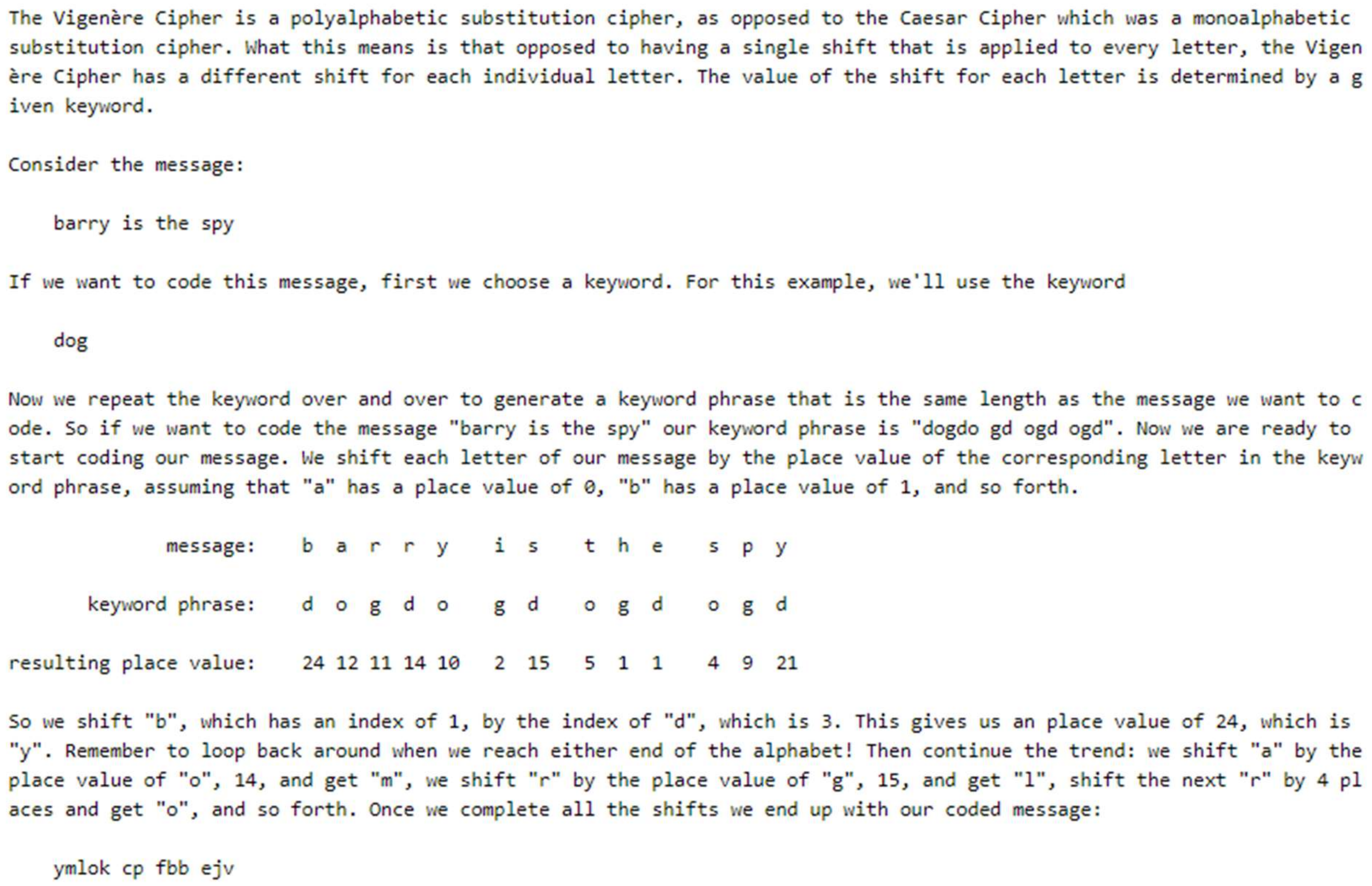
- I chose this method due to the low number of possible offsets. This makes the solution much easier to find and I therefore deemed it a more efficient method to obtaining the desired result.

Result



```
offset = 6:
bnlotsdqr gzud qdmcdqdc zkk ne sgdrd nkc bhogdqr narnkdsd. vd'kk gzud sn qdzkkx rsdo to ntq fzld he vd vzms sn jddo ntq ldrrzfd
r rzed.
offset = 7:
computers have rendered all of these old ciphers obsolete. we'll have to really step up our game if we want to keep our message
s safe.
offset = 8:
dpnqvufst ibwf sfoefsfe bmm pg uiftf pme djqifst pctpmfuf. xf'mm ibwf up sfbmmz tufq vq pvs hbnf jg xf xbou up lffq pvs nfttbhf
```

5. The Vigenere Chipher



The Vigenère Cipher is a polyalphabetic substitution cipher, as opposed to the Caesar Cipher which was a monoalphabetic substitution cipher. What this means is that opposed to having a single shift that is applied to every letter, the Vigenère Cipher has a different shift for each individual letter. The value of the shift for each letter is determined by a given keyword.

Consider the message:

    barry is the spy

If we want to code this message, first we choose a keyword. For this example, we'll use the keyword

    dog

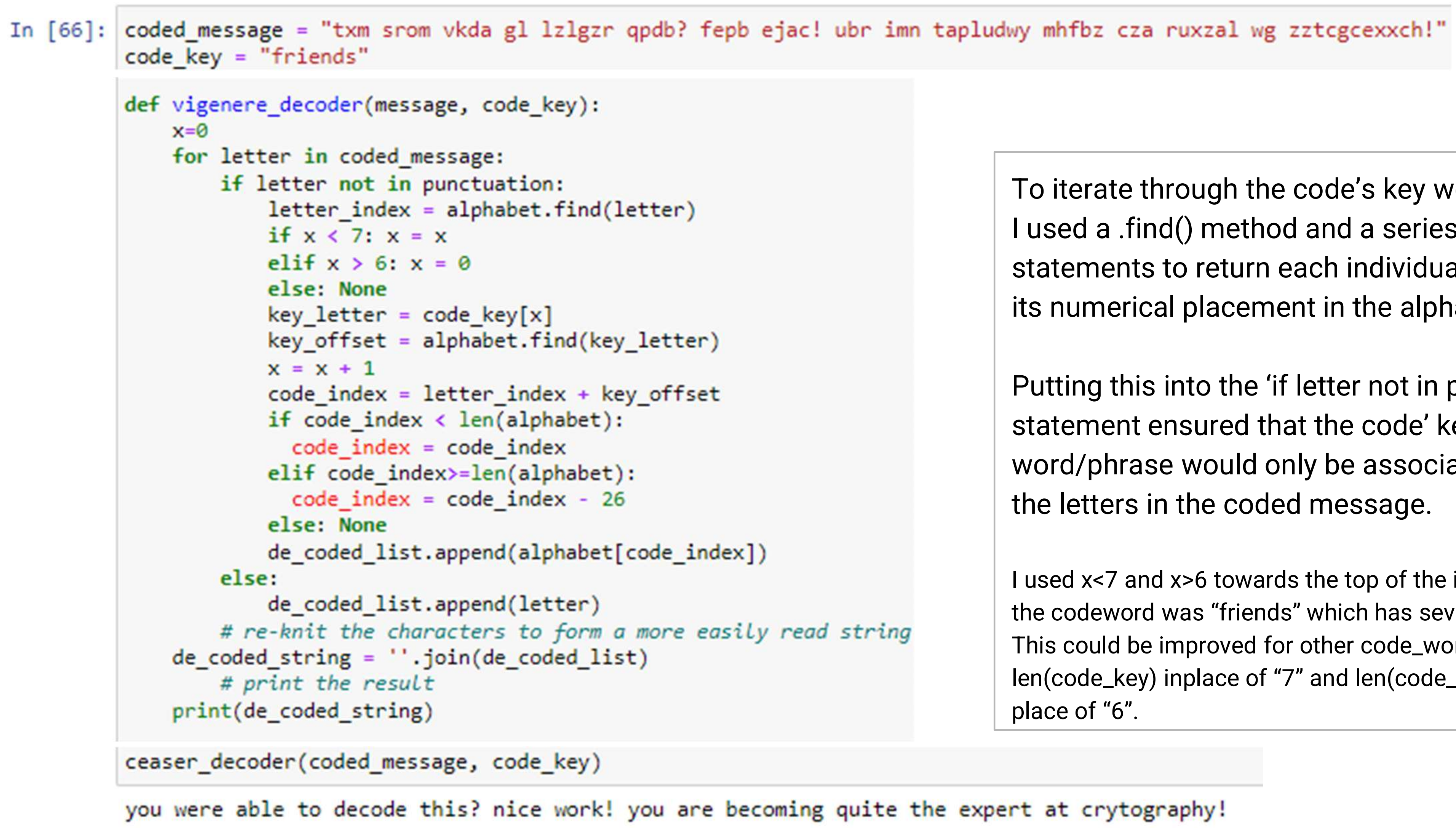
Now we repeat the keyword over and over to generate a keyword phrase that is the same length as the message we want to code. So if we want to code the message "barry is the spy" our keyword phrase is "dogdo gd ogd ogd". Now we are ready to start coding our message. We shift each letter of our message by the place value of the corresponding letter in the keyword phrase, assuming that "a" has a place value of 0, "b" has a place value of 1, and so forth.

```
              message:   b  a  r  r  y   i  s   t  h  e   s  p  y

       keyword phrase:   d  o  g  d  o   g  d   o  g  d   o  g  d

resulting place value:   24 12 11 14 10  2  15  5  1  1   4  9  21
```

So we shift "b", which has an index of 1, by the index of "d", which is 3. This gives us an place value of 24, which is "y". Remember to loop back around when we reach either end of the alphabet! Then continue the trend: we shift "a" by the place value of "o", 14, and get "m", we shift "r" by the place value of "g", 15, and get "l", shift the next "r" by 4 places and get "o", and so forth. Once we complete all the shifts we end up with our coded message:

    ymlok cp fbb ejv

5. Solving a Vigenère Cipher message.



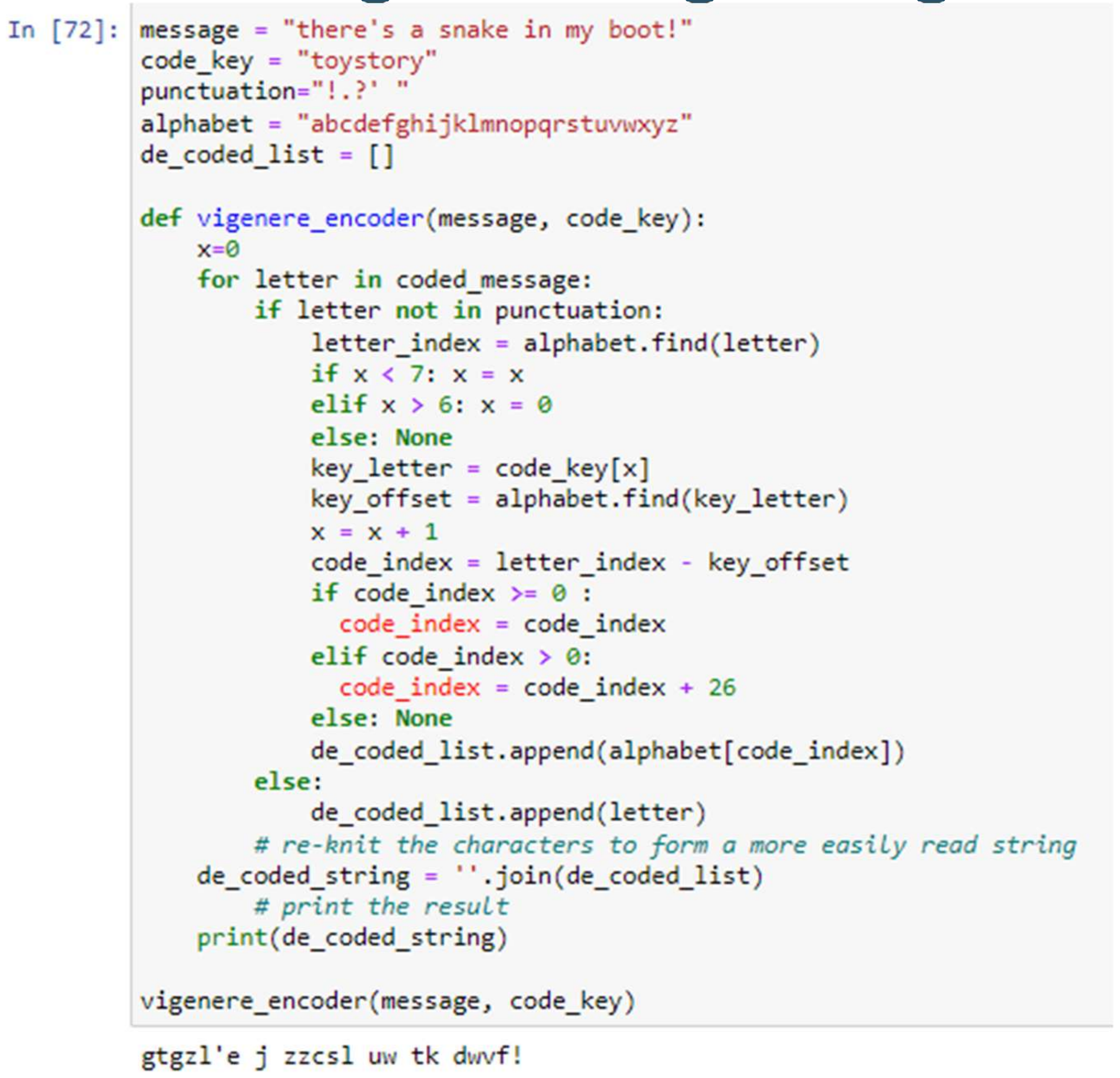
```
In [66]: coded_message = "txm srom vkda gl lzlgzr qpdb? fepb ejac! ubr imn tapludwy mhfbz cza ruxzal wg zztcgcexxch!"
         code_key = "friends"

         def vigenere_decoder(message, code_key):
             x=0
             for letter in coded_message:
                 if letter not in punctuation:
                     letter_index = alphabet.find(letter)
                     if x < 7: x = x
                     elif x > 6: x = 0
                     else: None
                     key_letter = code_key[x]
                     key_offset = alphabet.find(key_letter)
                     x = x + 1
                     code_index = letter_index + key_offset
                     if code_index < len(alphabet):
                       code_index = code_index
                     elif code_index>=len(alphabet):
                       code_index = code_index - 26
                     else: None
                     de_coded_list.append(alphabet[code_index])
                 else:
                     de_coded_list.append(letter)
                 # re-knit the characters to form a more easily read string
             de_coded_string = ''.join(de_coded_list)
                 # print the result
             print(de_coded_string)

         ceaser_decoder(coded_message, code_key)

         you were able to decode this? nice work! you are becoming quite the expert at crytography!
```

To iterate through the code's key word/phrase I used a .find() method and a series of if/elif statements to return each individual letter and its numerical placement in the alphabet.

Putting this into the 'if letter not in punctuation' statement ensured that the code' key word/phrase would only be associated with the letters in the coded message.

I used x<7 and x>6 towards the top of the if function as the codeword was "friends" which has seven letters. This could be improved for other code_words but using len(code_key) inplace of "7" and len(code_word)-1 in place of "6".

6. Encoding a message using the Vigenère Cipher.



```
In [72]: message = "there's a snake in my boot!"
         code_key = "toystory"
         punctuation="!.?' "
         alphabet = "abcdefghijklmnopqrstuvwxyz"
         de_coded_list = []

         def vigenere_encoder(message, code_key):
             x=0
             for letter in coded_message:
                 if letter not in punctuation:
                     letter_index = alphabet.find(letter)
                     if x < 7: x = x
                     elif x > 6: x = 0
                     else: None
                     key_letter = code_key[x]
                     key_offset = alphabet.find(key_letter)
                     x = x + 1
                     code_index = letter_index - key_offset
                     if code_index >= 0 :
                       code_index = code_index
                     elif code_index > 0:
                       code_index = code_index + 26
                     else: None
                     de_coded_list.append(alphabet[code_index])
                 else:
                     de_coded_list.append(letter)
                 # re-knit the characters to form a more easily read string
             de_coded_string = ''.join(de_coded_list)
                 # print the result
             print(de_coded_string)

         vigenere_encoder(message, code_key)

         gtgzl'e j zzcsl uw tk dwvf!
```

Altering the existing code from question 5 to encode messages rather than decode them.
This includes:
- the if/elif block of code to cycle through the key code word.
- the if/elif block of code to cycle through the alphabet in reverse order.