Casual Coded Correspondence: The Project

In this project, you will be working to code and decode various messages between you and your fictional cryptography enthusiast pen pal Vishal. You and Vishal have been exchanging letters for quite some time now and have started to provide a puzzle in each one of your letters. 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 Cipher. Here's how i
t works: You take your message, something like "hello" and then you shift all of the letters by a certain offset. For example, if I chose
 an offset of 3 and a message of "hello", I would code my message by shifting each letter 3 places to the left (with respect to the alphab
et). So "h" becomes "e", "e" becomes, "b", "l" becomes "i", and "o" becomes "l". Then I have my coded message,"ebiil"! Now I can send you
 my message and the offset and you can decode it. The best thing is that Julius Caesar himself used this cipher, that's why it's called th
e Caesar Cipher! Isn't that so cool! Okay, now I'm going to send you a longer coded 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?
```

Step 1: Decode Vishal's Message

In the cell below, use your Python skills to decode Vishal's message and print the result. Hint: you can account for shifts that go past the end of the alphabet using the modulus operator, but I'll let you figure out how!

In [33]:
```python
alphabets = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e', 6:'f', 7:'g', 8:'h', 9:'i', 10:'j', 11:'k', 12:'l',
             13:'m', 14:'n', 15:'o', 16:'p', 17:'q', 18:'r', 19:'s', 20:'t', 21:'u', 22:'v', 23:'w',
             24:'x', 25:'y', 26:'z'}

def decoder(sample_string):
    decoded_message = ""
    for i in sample_string:
        if i == " " or i == "." or i == '!' or i == "?":
            decoded_message += i

        for key, value in alphabets.items():
            if i == value:
                index = key + 10
                if index > 26:
                    decoded_message += alphabets[index - 26]

                else:
                    decoded_message += alphabets[index]

    return decoded_message

print(decoder('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!'))
```

```
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 offse
t!
```

Step 2: Send Vishal a Coded Message

Great job! Now send Vishal back a message using the same offset. Your message can be anything you want! Remember, coding happens in opposite direction of decoding.

In [34]:
```python
alphabets = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e', 6:'f', 7:'g', 8:'h', 9:'i', 10:'j', 11:'k', 12:'l',
             13:'m', 14:'n', 15:'o', 16:'p', 17:'q', 18:'r', 19:'s', 20:'t', 21:'u', 22:'v', 23:'w',
             24:'x', 25:'y', 26:'z'}

def encoder(sample_string):
    encoded_message = ""
    for i in sample_string:
        if i == " " or i == "." or i == "?" or i == "!":
            encoded_message += i

        for key, value in alphabets.items():
            if i == value:
                index = key - 10
                if index < 1:
                    encoded_message += alphabets[26 + index]
                else:
                    encoded_message += alphabets[index]
    return encoded_message

print(encoder('hey! i was able to decode it!'))
```

```
xuo! y mqi qrbu je tusetu yj!
```

Step 3: Make functions for decoding and coding

Vishal sent over another reply, this time with two coded messages!

```
You're getting the hang of this! Okay here are two more messages, the first one is coded just like before with  an offset of ten, and it c
ontains the hint for decoding the second message!

First message:

    jxu evviuj veh jxu iusedt cuiiqwu yi vekhjuud.

Second message:

    bqdradyuzs ygxfubxq omqemd oubtqde fa oapq kagd yqeemsqe ue qhqz yadq eqogdq!
```

Decode both of these messages.

If you haven't already, define two functions `decoder(message, offset)` and `coder(message, offset)` that can be used to quickly decode and code messages given any offset.

In [35]:
```python
alphabets = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e', 6:'f', 7:'g', 8:'h', 9:'i', 10:'j', 11:'k', 12:'l',
             13:'m', 14:'n', 15:'o', 16:'p', 17:'q', 18:'r', 19:'s', 20:'t', 21:'u', 22:'v', 23:'w',
             24:'x', 25:'y', 26:'z'}

def decoder(sample_string, offset):
    decoded_message = ""
    for i in sample_string:
        if i == " " or i == "." or i == '!' or i == "?":
            decoded_message += i

        for key, value in alphabets.items():
            if i == value:
                index = key + offset
                if index > 26:
                    decoded_message += alphabets[index - 26]

                else:
                    decoded_message += alphabets[index]

    return decoded_message

print(decoder('bqdradyuzs ygxfubxq omqemd oubtqde fa oapq kagd yqeemsqe ue qhqz yadq eqogdq!', 14))
```

```
performing multiple caesar ciphers to code your messages is even more secure!
```

In [36]:
```python
alphabets = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e', 6:'f', 7:'g', 8:'h', 9:'i', 10:'j', 11:'k', 12:'l',
             13:'m', 14:'n', 15:'o', 16:'p', 17:'q', 18:'r', 19:'s', 20:'t', 21:'u', 22:'v', 23:'w',
             24:'x', 25:'y', 26:'z'}

def encoder(sample_string, offset):
    encoded_message = ""
    for i in sample_string:
        if i == " " or i == "." or i == "?" or i == "!":
            encoded_message += i

        for key, value in alphabets.items():
            if i == value:
                index = key - offset
                if index < 1:
                    encoded_message += alphabets[26 + index]
                else:
                    encoded_message += alphabets[index]
    return encoded_message

print(encoder('hey! i was able to decode it!', 10))
```

```
xuo! y mqi qrbu je tusetu yj!
```

Step 4: Solving a Caesar Cipher without knowing the shift value

Awesome work! While you were working to decode his last two messages, Vishal sent over another letter! He's really been bitten by the crytpo-bug. Read it and see what interesting task he has lined up for you this time.

```
        Hello again friend! I knew you would love the Caesar Cipher, it's a cool simple way to encrypt messages. Did you know that back in
Caesar's time, it was considered a very secure way of communication and it took a lot of effort to crack if you were unaware of the value
 of the shift? That's all changed with computers! Now we can brute force these kinds of ciphers very quickly, as I'm sure you can imagine.

        To test your cryptography skills, this next coded message is going to be harder than the last couple to crack. It's still going to
be coded with a Caesar Cipher but this time I'm not going to tell you the value of   the shift. You'll have to brute force it yourself.

        Here's the coded message:

        vhfinmxkl atox kxgwxkxw tee hy maxlx hew vbiaxkl tl hulhexmx. px'ee atox mh kxteer lmxi ni hnk ztfx by px ptgm mh dxxi hnk fxlltzx
l ltyx.

        Good luck!
```

Decode Vishal's most recent message and see what it says!

In [37]:
```python
alphabets = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e', 6:'f', 7:'g', 8:'h', 9:'i', 10:'j', 11:'k', 12:'l', 13:'m',
             14:'n', 15:'o', 16:'p', 17:'q', 18:'r', 19:'s', 20:'t', 21:'u', 22:'v', 23:'w', 24:'x', 25:'y', 26:'z'}

def decoder(sample_string, offset):
    decoded_message = ""
    for i in sample_string:
        if i == " " or i == "." or i == '!' or i == "?" or i == "\'":
            decoded_message += i

        for key, value in alphabets.items():
            if i == value:
                index = key + offset
                if index > 26:
                    decoded_message += alphabets[index - 26]

                else:
                    decoded_message += alphabets[index]

    return decoded_message

for k in range(1, 26):
    print(decoder('vhfinmxkl atox kxgwxkxw tee hy maxlx hew vbiaxkl tl hulhexmx. px\'ee' +
                  ' atox mh kxteer lmxi ni hnk ztfx by px ptgm mh dxxi hnk fxlltzxl ltyx.', k))
```

```
wigjonylm bupy lyhxylyx uff iz nbymy ifx wcjbylm um ivmifyny. qy'ff bupy ni lyuffs mnyj oj iol augy cz qy quhn ni eyyj iol gymmuaym
muzy.
xjhkpozmn cvqz mziyzmzy vgg ja oczнz jgy xdkczmn vn jwnjgzoz. rz'gg cvqz oj mzvggt nozk pk jpm bvhz da rz rvio oj fzzk jpm hznnvbzn
nvaz.
ykilqpano dwra najzanaz whh kb pdaoa khz yeldano wo kxokhapa. sa'hh dwra pk nawhhu opal ql kqn cwia eb sa swjp pk gaal kqn iaoowcao
owba.
zljmrqbop exsb obkaboba xii lc qebpb lia zfmebop xp lyplibqb. tb'ii exsb ql obxiiv pqbm rm lro dxjb fc tb txkq ql hbbm lro jbppxdbp
pxcb.
amknsrcpq fytc pclbcpcb yjj md rfcqc mjb agnfcpq yq mzqmjcrc. uc'jj fytc rm pcyjjw qrcn sn msp eykc gd uc uylr rm iccn msp kcqqyecq
qydc.
bnlotsdqr gzud qdmcdqdc zkk ne sgdrd nkc bhogdqr zr narnkdsd. vd'kk gzud sn qdzkkx rsdo to ntq fzld he vd vzms sn jddo ntq ldrrzfdr
rzed.
computers have rendered all of these old ciphers as obsolete. we'll have to really step up our game if we want to keep our messages
safe.
dpnqvufst ibwf sfoefsfe bmm pg uiftf pme djqifst bt pctpmfuf. xf'mm ibwf up sfbmmz tufq vq pvs hbnf jg xf xbou up lffq pvs nfttbhft
tbgf.
eqorwvgtu jcxg tgpfgtgf cnn qh vjgug qnf ekrjgtu cu qduqngvg. yg'nn jcxg vq tgcnna uvgr wr qwt icog kh yg ycpv vq mggr qwt oguucigu
uchg.
frpsxwhuv kdyh uhqghuhg doo ri wkhvh rog flskhuv dv revrohwh. zh'oo kdyh wr uhdoob vwhs xs rxu jdph li zh zdqw wr nhhs rxu phvvdjhv
vdih.
gsqtyxivw lezi virhivih epp sj xliwi sph gmtlivw ew sfwspixi. ai'pp lezi xs vieppc wxit yt syv keqi mj ai aerx xs oiit syv qiwwekiw
weji.
htruzyjwx mfaj wjsijwji fqq tk ymjxj tqi hnumjwx fx tgxtqjyj. bj'qq mfaj yt wjfqqd xyju zu tzw lfrj nk bj bfsy yt pjju tzw rjxxfljx
xfkj.
iusvazkxy ngbk xktjkxkj grr ul znkyk urj iovnkxy gy uhyurkzk. ck'rr ngbk zu xkgrre yzkv av uax mgsk ol ck cgtz zu qkkv uax skyygmky
yglk.
jvtwbalyz ohcl yluklylk hss vm aolzl vsk jpwolyz hz vizvslal. dl'ss ohcl av ylhssf zalw bw vby nhtl pm dl dhua av rllw vby tlzzhnlz
zhml.
kwuxcbmza pidm zmvlmzml itt wn bpmam wtl kqxpmza ia wjawtmbm. em'tt pidm bw zmittg abmx cx wcz oium qn em eivb bw smmx wcz umaaioma
ainm.
lxvydcnab qjen anwmnanm juu xo cqnbn xum lryqnab jb xkbxuncn. fn'uu qjen cx anjuuh bcny dy xda pjvn ro fn fjwc cx tnny xda vnbbjpnb
bjon.
mywzedobc rkfo boxnobon kvv yp droco yvn mszrobc kc ylcyvodo. go'vv rkfo dy bokvvi cdoz ez yeb qkwo sp go gkxd dy uooz yeb wocckqoc
ckpo.
nzxafepcd slgp cpyopcpo lww zq espdp zwo ntaspcd ld zmdzwpep. hp'ww slgp ez cplwwj depa fa zfc rlxp tq hp hlye ez vppa zfc xpddlrpd
dlqp.
oaybgfqde tmhq dqzpqdqp mxx ar ftqeq axp oubtqde me aneaxqfq. iq'xx tmhq fa dqmxxk efqb gb agd smyq ur iq imzf fa wqqb agd yqeemsqe
emrq.
pbzchgref unir eraqrerq nyy bs gurfr byq pvcuref nf bofbyrgr. jr'yy unir gb ernyyl fgrc hc bhe tnzr vs jr jnag gb xrrc bhe zrffntrf
fnsr.
qcadihsfg vojs fsbrsfsr ozz ct hvsgs czr qwdvsfg og cpgczshs. ks'zz vojs hc fsozzm ghsd id cif uoas wt ks kobh hc yssd cif asggousg
gots.
rdbejitgh wpkt gtcstgts paa du iwtht das rxewtgh ph dqhdatit. lt'aa wpkt id gtpaan hite je djg vpbt xu lt lpci id ztte djg bthhpvth
hput.
secfkjuhi xqlu hudtuhut qbb ev jxuiu ebt syfxuhi qi eriebuju. mu'bb xqlu je huqbbo ijuf kf ekh wqcu yv mu mqdj je auuf ekh cuiiqwui
iqvu.
tfdglkvij yrmv iveuvivu rcc fw kyvjv fcu tzgyvij rj fsjfcvkv. nv'cc yrmv kf ivrccp jkvg lg fli xrdv zw nv nrek kf bvvg fli dvjjrxvj
jrwv.
ugehmlwjk zsnw jwfvwjwv sdd gx lzwkw gdv uahzwjk sk gtkgdwlw. ow'dd zsnw lg jwsddq klwh mh gmj ysew ax ow osfl lg cwwh gmj ewkksywk
ksxw.
```

Step 5: The Vigenère Cipher

Great work! While you were working on the brute force cracking of the cipher, Vishal sent over another letter. That guy is a letter machine!

```
         Salutations! As you can see, technology has made brute forcing simple ciphers like the Caesar Cipher extremely easy, and us crypto
    -enthusiasts have had to get more creative and use more complicated ciphers. This next cipher I'm going to teach you is the Vigenère Ciphe
    r, invented by an Italian cryptologist named Giovan Battista Bellaso (cool name eh?) in the 16th century, but named after another cryptolo
    gist from the 16th century, Blaise de Vigenère.

        The Vigenère Cipher is a polyalphabetic substitution cipher, as opposed to the Caesar Cipher which was a monoalphabetic substitutio
    n cipher. What this means is that opposed to having a single shift that is applied to every letter, the Vigenère Cipher has a different sh
    ift 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 use the repeat the keyword over and over to generate a _keyword phrase_ that is the same length as the message we want to co
    de. 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 the 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. Remember, we zero-index because this is Python we're talking about!

                 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:      4  14 15 12 16   24 11  21 25 22  22 17 5

         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 4, which is "e". Then c
    ontinue the trend: we shift "a" by the place value of "o", 14, and get "o" again, we shift "r" by the place value of "g", 15, and get "x",
    shift the next "r" by 12 places and "u", and so forth. Once we complete all the shifts we end up with our coded message:

             eoxum ov hnh gvb

         As you can imagine, this is a lot harder to crack without knowing the keyword! So now comes the hard part. I'll give you a message
    and the keyword, and you'll see if you can figure out how to crack it! Ready? Okay here's my message:

             dfc aruw fsti gr vjtwhr wznj? vmph otis! cbx swv jipreneo uhllj kpi rahjib eg fjdkwkedhmp!

         and the keyword to decode my message is

             friends

         Because that's what we are! Good luck friend!
```

And there it is. Vishal has given you quite the assignment this time! Try to decode his message. It may be helpful to create a function that takes two parameters, the coded message and the keyword and then work towards a solution from there.

NOTE: Watch out for spaces and punctuation! When there's a space or punctuation mark in the original message, there should be a space/punctuation mark in the corresponding repeated-keyword string as well!

```
In [38]: def vig_decoder(message, keyword):
             no_space_message = ""
             matching_string_keyword = ""
             decoded_message = ""

             alphabets = {'a':0, 'b':1, 'c':2, 'd':3, 'e':4, 'f':5, 'g':6,'h':7, 'i':8,'j':9, 'k':10, 'l':11,
                          'm':12, 'n':13, 'o':14, 'p':15, 'q':16, 'r':17, 's':18, 't':19, 'u':20,'v':21,'w':22,
                          'x':23,'y':24,'z':25}
             alphabets_other = {0:'a', 1:'b', 2:'c', 3:'d', 4:'e', 5:'f', 6:'g', 7:'h', 8:'i', 9:'j', 10:'k',
                                11:'l', 12:'m', 13:'n', 14:'o', 15:'p', 16:'q', 17:'r', 18:'s', 19:'t', 20:'u',
                                21:'v', 22:'w', 23:'x', 24:'y', 25:'z'}

             #Saving the original message without punctuation or spaces into a variable called no_space_message
             for character in message:
                 if character == " " or character == "?" or character == "!":
                     continue
                 else:
                     no_space_message += character

             #Creating a string with the keyword to match no_space_message. This string is saved in variable matching_string_keyword
             while len(matching_string_keyword) <= len(no_space_message):
                 matching_string_keyword += keyword

             #Due to extra characters in matching_string_keyword, the length of this string is being adjusted here
             updated_matching_string_keyword = matching_string_keyword[0:len(no_space_message)]

             #print(no_space_message)
             #print(updated_matching_string_keyword)


             #The decoded message is being generated here as per the program instructions - it is saved in the variable decoded_message
             for i in range(len(no_space_message)):
                 index = alphabets[no_space_message[i]] - alphabets[updated_matching_string_keyword[i]]

                 if index < 0:
                     index = 26 + index

                 decoded_message += alphabets_other[index]

             #print(message)
             #print(decoded_message)

             #Adding spaces and punctuation to the decoded message saved in decoded_message, which didn't have spaces and punctuation
             final_decoded_message = ""
             index = 0

             for k in message:
                 if k == " ":
                     final_decoded_message += " "
                 elif k == "?":
                     final_decoded_message += "?"
                 elif k == "!":
                     final_decoded_message += "!"
                 else:
                     final_decoded_message += decoded_message[index]
                     index += 1

             return final_decoded_message

         print(vig_decoder('dfc aruw fsti gr vjtwhr wznj? vmph otis! cbx swv jipreneo uhllj kpi rahjib eg fjdkwkedhmp!', 'friends'))
```

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

Step 6: Send a message with the Vigenère Cipher

Great work decoding the message. For your final task, write a function that can encode a message using a given keyword and write out a message to send to Vishal!

As a bonus, try calling your decoder function on the result of your encryption function. You should get the original message back!

In [39]:
```python
def vig_encoder(message, keyword):
    no_space_message = ""
    matching_string_keyword = ""
    encoded_message = ""

    alphabets = {'a':0, 'b':1, 'c':2, 'd':3, 'e':4, 'f':5, 'g':6,'h':7, 'i':8,'j':9, 'k':10, 'l':11,
                 'm':12, 'n':13, 'o':14, 'p':15, 'q':16, 'r':17, 's':18, 't':19, 'u':20,'v':21,'w':22,
                 'x':23,'y':24,'z':25}
    alphabets_other = {0:'a', 1:'b', 2:'c', 3:'d', 4:'e', 5:'f', 6:'g', 7:'h', 8:'i', 9:'j', 10:'k', 11:'l',
                       12:'m', 13:'n', 14:'o', 15:'p', 16:'q', 17:'r', 18:'s', 19:'t', 20:'u', 21:'v', 22:'w',
                       23:'x', 24:'y', 25:'z'}

    #Saving the original message without punctuation or spaces into a variable called no_space_message
    for character in message:
        if character == " " or character == "?" or character == "!":
            continue
        else:
            no_space_message += character

    #Creating a string with the keyword to match no_space_message. This string is saved in variable matching_string_keyword
    while len(matching_string_keyword) <= len(no_space_message):
        matching_string_keyword += keyword

    #Due to extra characters in matching_string_keyword, the length of this string is being adjusted here
    updated_matching_string_keyword = matching_string_keyword[0:len(no_space_message)]

    #print(no_space_message)
    #print(updated_matching_string_keyword)


    #The decoded message is being generated here as per the program instructions - it is saved in the variable decoded_message
    for i in range(len(no_space_message)):
        index = alphabets[no_space_message[i]] + alphabets[updated_matching_string_keyword[i]]

        if index > 25:
            index = index - 26

        encoded_message += alphabets_other[index]

    #print(message)
    #print(encoded_message)

    #Adding spaces and punctuation to the decoded message saved in decoded_message, which didn't have spaces and punctuation
    final_encoded_message = ""
    index = 0

    for k in message:
        if k == " ":
            final_encoded_message += " "
        elif k == "?":
            final_encoded_message += "?"
        elif k == "!":
            final_encoded_message += "!"
        else:
            final_encoded_message += encoded_message[index]
            index += 1

    return final_encoded_message

print(vig_encoder('you were able to decode this? nice work! you are becoming quite the expert at crytography!', 'friends'))
```

```
dfc aruw fsti gr vjtwhr wznj? vmph otis! cbx swv jipreneo uhllj kpi rahjib eg fjdkwkedhmp!
```

Conclusion

Over the course of this project you've learned about two different cipher methods and have used your Python skills to code and decode messages. There are all types of other facinating ciphers out there to explore, and Python is the perfect language to implement them with, so go exploring!