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 it 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 alphabet). 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 the 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 [1]:
```
import string
alphabet = string.ascii_lowercase
mod = len(alphabet)
print(alphabet)
```

```
abcdefghijklmnopqrstuvwxyz
```

In [2]:
```
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 myjx jxu iqcu evviuj!"""
offset = 10
```

In [3]:
```
encodes_message = []
for char in message:
    if char in alphabet:
        index = alphabet.find(char)
        encodes_message.append(alphabet[(index+offset)%mod])
    else:
        encodes_message.append(char)
```

In [4]:
```
encodes_message = ''.join(encodes_message)
print(encodes_message)
```

```
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 offset!
```

In [5]:
```
def decoder(message,offset):
    decoded_message = []
    for char in message:
        if char in alphabet:
            index = alphabet.find(char)
            decoded_message.append(alphabet[(index+offset)%mod])
        else:
            decoded_message.append(char)
    decoded_message = ''.join(decoded_message)
    return decoded_message
```

In [6]:
```
print(decoder(message,offset))
```

```
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 offset!
```

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 [7]:
```
def coder(message,offset):
    coded_message = []
    for char in message:
        if char in alphabet:
            index = alphabet.find(char)
            coded_message.append(alphabet[index-offset])
        else:
            coded_message.append(char)
    coded_message = ''.join(coded_message)
    return coded_message
```

In [8]:
```
mes1 = "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 offset!"
mes2 = "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!"

coder(mes1,offset) == mes2
```

Out[8]:
```
True
```

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 contains 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 [9]:
```
first_message = 'jxu evviuj veh jxu iusedt cuiiqwu yi vekhjuud.'
second_message = 'bqdradyuzs ygxfubxq omqemd oubtqde fa oapq kagd yqeemsqe ue qhqz yadq eqogdq!'
```

In [10]:
```
print(decoder(first_message,offset))
print()
print(decoder(second_message,14))
```

```
the offset for the second message is fourteen.

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

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 fxlltzxl ltyx.

Good luck!

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

In [11]:
```
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 fxlltzxl ltyx."

for i in range(mod):
    print("Offset={}".format(i))
    print(decoder(message,i))
    print()
```

```
Offset=0
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.

Offset=1
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.

Offset=2
xjhkpozmn cvqz mziyzmzy vgg ja ocznz jgy xdkczmn vn jwnjgzoz. rz'gg cvqz oj mzvggt nozk pk jpm bvhz da rz rvio oj fzzk jpm hznnvbzn nvaz.

Offset=3
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.

Offset=4
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.

Offset=5
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.

Offset=6
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.

Offset=7
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.

Offset=8
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.

Offset=9
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.

Offset=10
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.

Offset=11
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.

Offset=12
```

```
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.

Offset=13
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.

Offset=14
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.

Offset=15
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.

Offset=16
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.

Offset=17
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.

Offset=18
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.

Offset=19
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.

Offset=20
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.

Offset=21
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.

Offset=22
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.

Offset=23
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.

Offset=24
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.

Offset=25
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 Cipher, invented by an Italian cryptologist named Giovan Battista Bellaso (cool name eh?) in the 16th century, but named after
    another cryptologist 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
    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 use the 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 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
continue 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 [12]:
```
message = "dfc aruw fsti gr vjtwhr wznj? vmph otis! cbx swv jipreneo uhllj kpi rahjib eg fjdkwkedhmp!"
keyword = "friends"
```

In [13]:
```
def Vigenere_Cipher_decoder(message,keyword):
    len_of_message = len(message)
    while len(keyword) < len_of_message:
        keyword += keyword

    for i in range(len_of_message):
        if not message[i] in alphabet:
            keyword = keyword[:i]+message[i]+keyword[i:]
    keyword = keyword[:len_of_message]

    decoded_message = []
    for i in range(len_of_message):
        if message[i] in alphabet:
            index = alphabet.find(message[i])
            offset = alphabet.find(keyword[i])
            decoded_message.append(alphabet[(index-offset)%mod])
        else:
            decoded_message.append(message[i])

    decoded_message = ''.join(decoded_message)

    return decoded_message
```

In [14]:
```
print(Vigenere_Cipher_decoder(message,keyword))
```

```
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 [15]:
```
def Vigenere_Cipher_encoder(message,keyword):
    len_of_message = len(message)
    while len(keyword) < len_of_message:
        keyword += keyword

    for i in range(len_of_message):
        if not message[i] in alphabet:
            keyword = keyword[:i]+message[i]+keyword[i:]
    keyword = keyword[:len_of_message]
```

```
    encoded_message = []
    for i in range(len_of_message):
        if message[i] in alphabet:
            index = alphabet.find(message[i])
            offset = alphabet.find(keyword[i])
            encoded_message.append(alphabet[(index+offset)%mod])
        else:
            encoded_message.append(message[i])

    encoded_message = ''.join(encoded_message)

    return encoded_message
```

In [16]:

```
my_message = "hi, my name is martyna. i'm excited about machine learning. kisses!"
my_keyword = "konrad"

print(my_message)

encoded_message = Vigenere_Cipher_encoder(my_message,my_keyword)
print(encoded_message)

decoded_message = Vigenere_Cipher_decoder(encoded_message,my_keyword)
print(decoded_message)
```

```
hi, my name is martyna. i'm excited about machine learning. kisses!
rw, zp ndws vj mdbhlea. l'w sktiwor nsoxd anthlxs yvauxwax. klcgrj!
hi, my name is martyna. i'm excited about machine learning. kisses!
```

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!