JUNE 8TH VIGENÉRE BREAKING

I really hate this darn machine. I wish that they would sell it. It never does quite what I want. But only what I tell it.– Anonymous

We are going to write some functions in Sage that will help us break Vigenére ciphers.

First we need to figure out what the likely key length is. Remember, we do this with autocorrelation. First we write a function in Sage which takes ciphertext, shifts it over by a fixed amount, and counts how many times two letters in the same column are the same.

K	N	U	N	U	G	V	T	X	U	**T**	B	O	O	G	A	R	X	X	E	T	N	**E**	W	Q	N	M	W	E	L
			K	N	U	N	U	G	V	**T**	X	U	T	B	O	O	G	A	R	X	X	**E**	T	N	E	W	Q	N	M

For instance, there are two such places in the above text, at the bolded **T** and **E**. We are going to write a function which counts how incidences you have for a given shift. In other words

`incidenceCount('KNUNUGVTXUTBOOGARXXETNEWQNMWEL',3)`

should output the number `2`.

I wrote my function like this (fill in the `...` with an appropriate loop).

```
def incidenceCount(s, n):
    counter = 0
    ...
    print str(n) + ": " + str(counter)
```

The reason we wrote this function is because if you have a keylength of n, then the incidence count at n, $2n$, $3n$ etc. will tend to be more than other incidence counts. Hence I wrote a function that calls incidenceCount over and over in a loop with different shifts, 2, 3, ..., up to a max shift of `maxInt`.

```
def autoCorrelation(s,maxInt):
    ...
```

When I ran `autoCorrelation(myStr, 12)` on the ciphertext `myStr` below.

```
VHXF IYHE KGNM CCVK DXPT LQFE KFXC RXPO MUOV JAGI ETDL XCSM JEYG EEKN ZUOY JUFC NGCT
NTEB JAWY OKME WJAK FFHT NXCR EATP QYXC RLHO KVHX UOEG PNTP HUEH HIGH ULKN ZNIY GIGV
OTPI GCNB OAMG BHFY YQRM JILK HTFD XRRB XEWO YLGL YQFK GSMC NWJE TNTA KHTF DXUI KGDB
VWBV HTPA KFON TTAC TYCR XZCX GDXF MHFE KCTB QNUW TGQW MJAM KHTF FBPI LJEW VHXD ETWT
RQFM JEWT ETOV TPIL JEWC NWDR XCTA NELU HHTR HTAG FDBU GNUT YKLE GDFA HXCR MWNT DLXV
OXPD NTEM JETU PXET HHTA GBXK NZKH TFCK GAMG DBTU LJEW QUMQ FMJE KQOF CNWE OGVI GWEW
CLHP GMKM XVRT XEKU IGIM RDEW EHTO BXTU GCBE GTHE OFRO LGMR OIGF THUL XGPT VLXP GMJL
TUSB VUWG SNEC XGDX FTHV HXVU FWLM KHTF BXHO KGEG FUKG DTPD BVHK GWFA SXNF HPTA GBXF
IGOY VNOM JELG NWGA OQUK KNZV OLGE DCFX YMHO EGVS HHFH TGXV FNNN XUSU WTBV WTUI GXAB
PILN EIVI GFEX FBNV IPCS WKSM WRUG DUAT AGWB NDXU TWTE TOSB VHHW GAVI LCWX NISC BXVH
BPTA GBEQ OFQF AGAE VHPC LDKN ZKNM JELV RXGT LQFB PGHN SMCD M
```

I obtained the output:

```
2: 22
3: 44
4: 26
5: 17
6: 37
7: 24
8: 23
9: 47
10: 23
11: 23
12: 40
```

Notice that I have bigger incidence counts for shifts of 3, 6, 9, 12. This means that the key length is probably 3 (and this is correct).

1. Write your own functions for `incidentCount(s,n)` and `autoCorrelation(s,maxInt)`.

Now that we know the key length, we need to find the key. This is done with simple frequency analysis. Note that if we encrypt with a key of length 3, then the 1st, 4th, 7th, 10th, etc. letter will all have been shifted by the same amount. Likewise for the 2nd, 5th, 8th, 11th etc. and the 3rd, 6th, 9th, 12th, etc. We are going to write a function that figures out what letters are most common in the 1st, 4th, 7th, etc. spots, and in the 2nd, 5th, 8th, etc spots, and so on.

I created a function that would take a string `s`, an expected key length `k` and a number `t` between `0` and `k`. It will find out how many of each letter are in the `t`th spot, the `(t + k)`th spot, the `(t + 2k)`th spot and so on. You can then use that to guess the original key and thus decrypt the message.

I wrote my function something like this. Again, fill in your own ....

```
#my function assumed only capital letters, and ignored spaces and punctuation.
def VigenereFrequency(s, k, t):
    letterList = [0]*26  #creates a list with 26 zeros
    while (t < len(s)):  #now fill in the letter list
        ...
    for i in range(0,26):  #finally display the letter list
        ...
```

2. Finish writing the function `VigenereFrequency(s, k, t)`.

3. Use your code to deduce the key size and then the key of the above encrypted message (it is a common English word).

4. Write a function `unVigenere(message,key)` that takes an encrypted message and the keyword used to encrypt it and returns the deciphered message.

5. Finally, use your code to decrypt the ciphertext available at the following address:

`http://www.math.utah.edu/~schwede/Camp2016/VigenereDecryptable.txt`

BONUS QUESTIONS

(1) Improve the functions you've written today. In particular, can you make your functions automatically decrypt things or at least reduce how much guesswork you have to do?
(2) A number is called a palindrome if the order of digits is the same when read backward or forward. For example, 101, 333, 123454321 and 3443 are palindromes. Write a function that finds all palindrom numbers smaller than 10,000.
(3) The number, 197, is called a circular prime because all rotations of the digits: 197, 971, and 719, are themselves prime.

    There are thirteen such primes below 100: 2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, and 97. Write a function that finds all circular primes below 1000.
(4) Notice that the multiplication equation $39 \cdot 186 = 7254$ uses each digit 1 through 9 exactly once. Write a function that finds all multiplication equations that use each digit 1 through 9 exactly once.