Written problems

1. The following passage has been encrypted using Caesar encryption. That is, there is a secret key k, and each letter of the plaintext has been advanced k places in the alphabet (where the letter Z advanced to the letter A). All punctuation and spaces have been left unchanged.

```
Nzky kyv yvcg fw kyv arezkfi yv jtivnvu fekf kyv
jzuv fw kyv uvjb r gvetzc jyrigvevi -- kyrk yzxycp
jrkzjwpzex, yzxycp gyzcfjfgyztrc zdgcvdvek kyrk xfvj
kztfeuvifxr-kztfeuvifxr, wvvuzex fe kyv pvccfn wzezjy reu
jnvvk nffu, reu veuj lg ze r bzeu fw jfleucvjjcp jgzeezex
vkyvivrc mfzu rj nv rcc dljk.
```

Determine the secret key k. (You can solve this by hand, but I suggest trying to write some code to make your job easier. If you can fully automate the process, you may submit your work for extra credit; see the last problem on this problem set.)

Solution.

The following bit of code implements the Caesar cipher.

```
# Rotate a character ch n places, preserving the case.
# If ch isn't a letter, then this function returns it unchanged.
def rotate_char(ch,n):
        if not(ch.isalpha()): return ch #If ch isn't a letter, don't change it.
        #Calculate the position in the alphabet (A is 0, B is 1, etc.)
        loc = ord(ch.upper()) - ord('A')
        #Advance loc by n places, wrapping if necessary
        loc = (loc + n) % 26
        #Return the new letter, in the appropriate case
        if ch.isupper(): return chr( ord('A') + loc )
        else: return chr( ord('a') + loc )

# Rotate an entire string by applying rotate_char to each letter
def rotate(line,n):
        res = ""
        for c in line:
                res += rotate_char(c,n)
        return res
# NOTE: the more "pythonic" one-liner for this function is:
#         return ''.join(rotate_char(ch,n) for ch in line)
```

Now, to quickly print all possible deciperings of this passage, you can first write the original passage to a string `s` and write a loop as follows (output shown, truncated after 20 characters

to save space). Note that I rotate `s` $-k$ places since I am deciphering, rather than enciphering, it.

```
>>> for k in range(26):
...     print n, rotate(s,-k)[:60], "..."
...
0 Nzky kyv yvcg fw kyv arezkfi yv jtivnvu fekf kyv jzuv fw kyv ...
1 Myjx jxu xubf ev jxu zqdyjeh xu ishumut edje jxu iytu ev jxu ...
2 Lxiw iwt wtae du iwt ypcxidg wt hrgtlts dcid iwt hxst du iwt ...
3 Kwhv hvs vszd ct hvs xobwhcf vs gqfsksr cbhc hvs gwrs ct hvs ...
4 Jvgu gur uryc bs gur wnavgbe ur fperjrq bagb gur fvqr bs gur ...
5 Iuft ftq tqxb ar ftq vmzufad tq eodqiqp azfa ftq eupq ar ftq ...
6 Htes esp spwa zq esp ulytezc sp dncphpo zyez esp dtop zq esp ...
7 Gsdr dro rovz yp dro tkxsdyb ro cmbogon yxdy dro csno yp dro ...
8 Frcq cqn qnuy xo cqn sjwrcxa qn blanfnm xwcx cqn brmn xo cqn ...
9 Eqbp bpm pmtx wn bpm rivqbwz pm akzmeml wvbw bpm aqlm wn bpm ...
10 Dpao aol olsw vm aol qhupavy ol zjyldlk vuav aol zpkl vm aol ...
11 Cozn znk nkrv ul znk pgtozux nk yixkckj utzu znk yojk ul znk ...
12 Bnym ymj mjqu tk ymj ofsnytw mj xhwjbji tsyt ymj xnij tk ymj ...
13 Amxl xli lipt sj xli nermxsv li wgviaih srxs xli wmhi sj xli ...
14 Zlwk wkh khos ri wkh mdqlwru kh vfuhzhg rqwr wkh vlgh ri wkh ...
15 Ykvj vjg jgnr qh vjg lcpkvqt jg uetgygf qpvq vjg ukfg qh vjg ...
16 Xjui uif ifmq pg uif kbojups if tdsfxfe poup uif tjef pg uif ...
17 With the help of the janitor he screwed onto the side of the ...
18 Vhsg sgd gdko ne sgd izmhsnq gd rbqdvdc nmsn sgd rhcd ne sgd ...
19 Ugrf rfc fcjn md rfc hylgrmp fc qapcucb mlrm rfc qgbc md rfc ...
20 Tfqe qeb ebim lc qeb gxkfqlo eb pzobtba lkql qeb pfab lc qeb ...
21 Sepd pda dahl kb pda fwjepkn da oynasaz kjpk pda oeza kb pda ...
22 Rdoc ocz czgk ja ocz evidojm cz nxmzrzy jioj ocz ndyz ja ocz ...
23 Qcnb nby byfj iz nby duhcnil by mwlyqyx ihni nby mcxy iz nby ...
24 Pbma max axei hy max ctgbmhk ax lvkxpxw hgmh max lbwx hy max ...
25 Oalz lzw zwdh gx lzw bsfalgj zw kujwowv gflg lzw kavw gx lzw ...
```

From this list, it is easy to pick out the one in English. The secret key used was $\boxed{\text{k=17}}$, and the plaintext is the following passage from the novel *Pnin*.

With the help of the janitor he screwed onto the side of the desk a pencil sharpener – that highly satisfying, highly philosophical implement that goes ticonderoga-ticonderoga, feeding on the yellow finish and sweet wood, and ends up in a kind of soundlessly spinning ethereal void as we all must.

In problem 8 I will show how to add another function to the code above that can pick out this plaintext automatically.

2. Throughout this course, we will say that an integer a is an "n-bit (nonnegative) integer" if $0 \leq a < 2^n$. This means that it can be written in binary with at most n symbols; it is a convenient shorthand to specify the size of a number, such as a cryptographic key. The number of bits in a key is a rough guide to its strength.

(a) The secret key of a Caesar cipher is a number k from 1 to 25 inclusive. How many bits are needed to specify this secret key?

Solution Since $2^5 = 32$, $\boxed{5 \text{ bits}}$ are enough to represent a number from 0 to 31, and hence sufficient for a Caesar cipher key.

(b) The *Data Encryption Standard* (DES) is a private-key encryption algorithm that was a government standard from 1977 to 2002. DES uses 56-bit secret keys. Suppose that Eve attempts a brute-force attack on DES by trying to decrypt an intercepted cipher text with every possible 56-bit key until she finds something that looks like English text. If Eve's system can try 1 billion keys per second, how long would it take her to try all of the keys (and thus be sure to break the encryption)?

(By 1999, a distributed system was able to break DES encryption in less than 24 hours. DES was replaced in 2002 by a new standard, called AES, which uses keys of at least 128 bits. For "top secret" communication, the government uses AES with 256 bit keys.)

Solution. Eve requires $2^{56}/10^9$ seconds to try all possible keys. This comes out to

$$2^{56}\frac{1}{10^9}\text{sec} \cdot \frac{\text{hour}}{3600\text{sec}} \cdot \frac{\text{day}}{24\text{hour}} \cdot \frac{\text{year}}{365.25\text{day}} \approx \boxed{2.28 \text{ years}}$$

This is a while, but short enough that you can see why it was necessary to move to standard with longer keys.

(c) A stronger (but still weak) version of the Caesar cipher is a *substitution cipher,* as described in §1.1. A secret key for a substitution cipher consists of a permutation of the letters from A to Z (see the book for some examples). How many such secret keys are there? How many bits would be required to specify such a key?

(Although substitution ciphers have fairly long keys, they are still completely insecure; the textbook describes how they can be broken using methods much more efficient than brute force search.)

Solution. The number of possible keys is equal to 26! (26 factorial), i.e. $26 \cdot 25 \cdot 24 \cdots 2 \cdot 1$. This is because there are 26 choices for where `A` is sent, then 25 choices (all but the letter taken for `A`) for where `B` is sent, and so on.

The number 26! is approximately $4.03 \cdot 10^{26}$. Its logarithm base 2 is $\log_2(26!) \approx 88.4$. Therefore at least $\boxed{89 \text{ bits}}$ are needed in order to represent the secret key of a substitution cipher (in practice, it might take more than this, since it is not obvious to associate these permutations with the numbers from 1 to 26!).

3. Textbook exercise 1.9. (same in first edition). If you wish, you may complete problem 7 first and run your code to answer this question.

Solution. The following lists of steps were generated automatically using the Euclidean algorithm (some of them run off the end of the page, but you can see the coefficients used).

(a)

$$\begin{aligned}
\gcd(291, 252) &= \gcd(252, 39) \quad (\text{since } 291 - (1) \cdot 252 = 39) \\
&= \gcd(39, 18) \quad (\text{since } 252 - (6) \cdot 39 = 18) \\
&= \gcd(18, 3) \quad (\text{since } 39 - (2) \cdot 18 = 3) \\
&= \gcd(3, 0) \quad (\text{since } 18 - (6) \cdot 3 = 0) \\
&= 3
\end{aligned}$$

(b)

$$\begin{aligned}
\gcd(16261, 85652) &= \gcd(85652, 16261) \quad (\text{since } 16261 - (0) \cdot 85652 = 16261) \\
&= \gcd(16261, 4347) \quad (\text{since } 85652 - (5) \cdot 16261 = 4347) \\
&= \gcd(4347, 3220) \quad (\text{since } 16261 - (3) \cdot 4347 = 3220) \\
&= \gcd(3220, 1127) \quad (\text{since } 4347 - (1) \cdot 3220 = 1127) \\
&= \gcd(1127, 966) \quad (\text{since } 3220 - (2) \cdot 1127 = 966) \\
&= \gcd(966, 161) \quad (\text{since } 1127 - (1) \cdot 966 = 161) \\
&= \gcd(161, 0) \quad (\text{since } 966 - (6) \cdot 161 = 0) \\
&= 161
\end{aligned}$$

(c)

$$\begin{aligned}
\gcd(139024789, 93278890) &= \gcd(93278890, 45745899) \\
&\quad (\text{since } 139024789 - (1) \cdot 93278890 = 45745899) \\
&= \gcd(45745899, 1787092) \quad (\text{since } 93278890 - (2) \cdot 45745899 = 1787092) \\
&= \gcd(1787092, 1068599) \quad (\text{since } 45745899 - (25) \cdot 1787092 = 1068599) \\
&= \gcd(1068599, 718493) \quad (\text{since } 1787092 - (1) \cdot 1068599 = 718493) \\
&= \gcd(718493, 350106) \quad (\text{since } 1068599 - (1) \cdot 718493 = 350106) \\
&= \gcd(350106, 18281) \quad (\text{since } 718493 - (2) \cdot 350106 = 18281) \\
&= \gcd(18281, 2767) \quad (\text{since } 350106 - (19) \cdot 18281 = 2767) \\
&= \gcd(2767, 1679) \quad (\text{since } 18281 - (6) \cdot 2767 = 1679) \\
&= \gcd(1679, 1088) \quad (\text{since } 2767 - (1) \cdot 1679 = 1088) \\
&= \gcd(1088, 591) \quad (\text{since } 1679 - (1) \cdot 1088 = 591) \\
&= \gcd(591, 497) \quad (\text{since } 1088 - (1) \cdot 591 = 497) \\
&= \gcd(497, 94) \quad (\text{since } 591 - (1) \cdot 497 = 94) \\
&= \gcd(94, 27) \quad (\text{since } 497 - (5) \cdot 94 = 27) \\
&= \gcd(27, 13) \quad (\text{since } 94 - (3) \cdot 27 = 13) \\
&= \gcd(13, 1) \quad (\text{since } 27 - (2) \cdot 13 = 1) \\
&= \gcd(1, 0) \quad (\text{since } 13 - (13) \cdot 1 = 0) \\
&= 1
\end{aligned}$$

(d)

$$\begin{aligned}
\gcd(16534528044, 8332745927) &= \gcd(8332745927, 8201782117) \text{ (since } 16534528044 - (1) \cdot 83327459 \\
&= \gcd(8201782117, 130963810) \text{ (since } 8332745927 - (1) \cdot 8201782117 \\
&= \gcd(130963810, 82025897) \text{ (since } 8201782117 - (62) \cdot 130963810 = \\
&= \gcd(82025897, 48937913) \text{ (since } 130963810 - (1) \cdot 82025897 = 489 \\
&= \gcd(48937913, 33087984) \text{ (since } 82025897 - (1) \cdot 48937913 = 3308 \\
&= \gcd(33087984, 15849929) \text{ (since } 48937913 - (1) \cdot 33087984 = 1584 \\
&= \gcd(15849929, 1388126) \text{ (since } 33087984 - (2) \cdot 15849929 = 13881 \\
&= \gcd(1388126, 580543) \text{ (since } 15849929 - (11) \cdot 1388126 = 580543) \\
&= \gcd(580543, 227040) \text{ (since } 1388126 - (2) \cdot 580543 = 227040) \\
&= \gcd(227040, 126463) \text{ (since } 580543 - (2) \cdot 227040 = 126463) \\
&= \gcd(126463, 100577) \text{ (since } 227040 - (1) \cdot 126463 = 100577) \\
&= \gcd(100577, 25886) \text{ (since } 126463 - (1) \cdot 100577 = 25886) \\
&= \gcd(25886, 22919) \text{ (since } 100577 - (3) \cdot 25886 = 22919) \\
&= \gcd(22919, 2967) \text{ (since } 25886 - (1) \cdot 22919 = 2967) \\
&= \gcd(2967, 2150) \text{ (since } 22919 - (7) \cdot 2967 = 2150) \\
&= \gcd(2150, 817) \text{ (since } 2967 - (1) \cdot 2150 = 817) \\
&= \gcd(817, 516) \text{ (since } 2150 - (2) \cdot 817 = 516) \\
&= \gcd(516, 301) \text{ (since } 817 - (1) \cdot 516 = 301) \\
&= \gcd(301, 215) \text{ (since } 516 - (1) \cdot 301 = 215) \\
&= \gcd(215, 86) \text{ (since } 301 - (1) \cdot 215 = 86) \\
&= \gcd(86, 43) \text{ (since } 215 - (2) \cdot 86 = 43) \\
&= \gcd(43, 0) \text{ (since } 86 - (2) \cdot 43 = 0) \\
&= 43
\end{aligned}$$

4. Textbook exercise 1.10 (same in first edition).

   Solution. Keeping track of the linear combinations giving rise to each intermediate number from the previous problem, we obtain the following (again, this output has been automatically generated).

   (a)

$$\begin{aligned}
291 &= 1 \cdot 291 + 0 \cdot 252 \\
252 &= 0 \cdot 291 + 1 \cdot 252 \\
39 &= 1 \cdot 291 - 1 \cdot 252 \\
18 &= -6 \cdot 291 + 7 \cdot 252 \\
3 &= 13 \cdot 291 - 15 \cdot 252 \\
0 &= -84 \cdot 291 + 97 \cdot 252
\end{aligned}$$

   So one possible answer is $\boxed{\text{(13,-15).}}$

(b)

$$\begin{aligned}
16261 &= 1 \cdot 16261 + 0 \cdot 85652 \\
85652 &= 0 \cdot 16261 + 1 \cdot 85652 \\
16261 &= 1 \cdot 16261 + 0 \cdot 85652 \\
4347 &= -5 \cdot 16261 + 1 \cdot 85652 \\
3220 &= 16 \cdot 16261 - 3 \cdot 85652 \\
1127 &= -21 \cdot 16261 + 4 \cdot 85652 \\
966 &= 58 \cdot 16261 - 11 \cdot 85652 \\
161 &= -79 \cdot 16261 + 15 \cdot 85652 \\
0 &= 532 \cdot 16261 - 101 \cdot 85652
\end{aligned}$$

So one possible answer is $\boxed{\text{(-79,15).}}$ You may notice an odd artifact at the beginning of these steps: the third line is identical to the first. This is because the smaller of the two numbers $16261, 85652$ was listed first, so the first thing the algorithm did (in effect) was swap the two values. You also see this occurring in the previous problem.

(c)

$$\begin{aligned}
139024789 &= 1 \cdot 139024789 + 0 \cdot 93278890 \\
93278890 &= 0 \cdot 139024789 + 1 \cdot 93278890 \\
45745899 &= 1 \cdot 139024789 - 1 \cdot 93278890 \\
1787092 &= -2 \cdot 139024789 + 3 \cdot 93278890 \\
1068599 &= 51 \cdot 139024789 - 76 \cdot 93278890 \\
718493 &= -53 \cdot 139024789 + 79 \cdot 93278890 \\
350106 &= 104 \cdot 139024789 - 155 \cdot 93278890 \\
18281 &= -261 \cdot 139024789 + 389 \cdot 93278890 \\
2767 &= 5063 \cdot 139024789 - 7546 \cdot 93278890 \\
1679 &= -30639 \cdot 139024789 + 45665 \cdot 93278890 \\
1088 &= 35702 \cdot 139024789 - 53211 \cdot 93278890 \\
591 &= -66341 \cdot 139024789 + 98876 \cdot 93278890 \\
497 &= 102043 \cdot 139024789 - 152087 \cdot 93278890 \\
94 &= -168384 \cdot 139024789 + 250963 \cdot 93278890 \\
27 &= 943963 \cdot 139024789 - 1406902 \cdot 93278890 \\
13 &= -3000273 \cdot 139024789 + 4471669 \cdot 93278890 \\
1 &= 6944509 \cdot 139024789 - 10350240 \cdot 93278890 \\
0 &= -93278890 \cdot 139024789 + 139024789 \cdot 93278890
\end{aligned}$$

So one possible answer is $\boxed{(6944509, -10350240).}$

(d)

$$
\begin{aligned}
16534528044 &= 1 \cdot 16534528044 + 0 \cdot 8332745927 \\
8332745927 &= 0 \cdot 16534528044 + 1 \cdot 8332745927 \\
8201782117 &= 1 \cdot 16534528044 - 1 \cdot 8332745927 \\
130963810 &= -1 \cdot 16534528044 + 2 \cdot 8332745927 \\
82025897 &= 63 \cdot 16534528044 - 125 \cdot 8332745927 \\
48937913 &= -64 \cdot 16534528044 + 127 \cdot 8332745927 \\
33087984 &= 127 \cdot 16534528044 - 252 \cdot 8332745927 \\
15849929 &= -191 \cdot 16534528044 + 379 \cdot 8332745927 \\
1388126 &= 509 \cdot 16534528044 - 1010 \cdot 8332745927 \\
580543 &= -5790 \cdot 16534528044 + 11489 \cdot 8332745927 \\
227040 &= 12089 \cdot 16534528044 - 23988 \cdot 8332745927 \\
126463 &= -29968 \cdot 16534528044 + 59465 \cdot 8332745927 \\
100577 &= 42057 \cdot 16534528044 - 83453 \cdot 8332745927 \\
25886 &= -72025 \cdot 16534528044 + 142918 \cdot 8332745927 \\
22919 &= 258132 \cdot 16534528044 - 512207 \cdot 8332745927 \\
2967 &= -330157 \cdot 16534528044 + 655125 \cdot 8332745927 \\
2150 &= 2569231 \cdot 16534528044 - 5098082 \cdot 8332745927 \\
817 &= -2899388 \cdot 16534528044 + 5753207 \cdot 8332745927 \\
516 &= 8368007 \cdot 16534528044 - 16604496 \cdot 8332745927 \\
301 &= -11267395 \cdot 16534528044 + 22357703 \cdot 8332745927 \\
215 &= 19635402 \cdot 16534528044 - 38962199 \cdot 8332745927 \\
86 &= -30902797 \cdot 16534528044 + 61319902 \cdot 8332745927 \\
43 &= 81440996 \cdot 16534528044 - 161602003 \cdot 8332745927 \\
0 &= -193784789 \cdot 16534528044 + 384523908 \cdot 8332745927
\end{aligned}
$$

So one possible solution is $\boxed{(81440996, -161602003).}$

5. Textbook exercise 1.11 parts (a) and (b) (same in first edition).

Solution.

(a) Suppose that $au + bv = 1$, where a, u, b, v are all integers. Let g be the greatest common divisor of a and b. Then we can write

$$1 = g \cdot \left(\frac{a}{g} u + \frac{b}{g} v \right).$$

Since a/g and b/g are both integers, it follows that 1 is equal to g times an integer, i.e. $g|1$. The only positive integer than divides 1 is 1 itself, hence $g = 1$, as desired.

(b) No, it is not necessarily true. For example, if $a = 3$ and $b = 2$, then we can take $u = 6, v = -6$ and obtain $au + bv = 6$, even though $\gcd(3, 2) = 1$.

What we can say for sure, following the logic of the previous problem, is that if $au + bv = 6$, then

$$6 = g \cdot \left(\frac{a}{g} u + \frac{b}{g} v \right),$$

where $g = \gcd(a, b)$ as before. Therefore certainly $\boxed{g|6.}$ In fact, this is the the most that we can say, as the converse is true: if $\gcd(a, b)$ divides 6, then 6 can be written $au + bv$.

To see this, simple observe that we can first write $g = au' + bv'$, and multiply both sides by $6/g$ (an integer) to obtain 6 as a combination of a and b.

Therefore the possible values of $\gcd(a, b)$, given that $6 = au + bv$ for some u and v, are $\boxed{1, 2, 3, \text{ and } 6.}$

Programming problems

6. Write a program which can factor a 16-bit integer into primes. More precisely, the program should take an integer n such that $2 \leq n < 2^{16}$, and print out the prime factors of n, in order, printing each prime the same number of times as it occurs in the factorization of n. For example, if the program reads "12", it should print "2 2 3."

   (Later in the course, you will write programs that can factor integers much longer than 16 bits.)

   Solution. For a 16-bit integer n, we can use a trial and error process: if we simply list every prime number p from 1 to n, we can attempt to divide n by p, and print p to the screen if p divides n. To get the multiplicity right, we can repeatedly replace n by n/p until it is no longer divisible by p (printing p each time).

   In fact, a further simplification is possible: we can do the above procedure, but list *every* integer p from 1 to n, without sifting out the primes. The reason that this will still work is that any time we find a divisor p of n during this process, we can be sure that if p had a proper factor d, then we must have *already tried* d, and divided n by it as many times as possible, so by this time $d \nmid n$. But since $d|p$ and $p|n$, this is impossible. This contradiction leads to the conclusion that, in the process described, any time we find a p dividing n, then p is necessarily prime.

   This strategy can be implemented in a few lines as follows.

```
def factor(n):
        p = 2
        while n>1:
                while (n%p == 0):
                        print p,
                        n /= p
                p += 1

n = int(raw_input())
factor(n)
```

7. Write a program which takes a list of 1024-bit positive integers (i.e. each integer a in the list satisfies $1 \leq a < 2^{1024}$) and prints their greatest common divisor.

   Suggestion. First write a function to compute the greatest common divisor of two positive integers (we will discuss a classic, very efficient, algorithm for this in class), then figure out how to use this function to find the GCD of a longer list.

Solution. First implement the `gcd` function using the Euclidean algorithm. Then, we can begin the first element in the list and successively "shrink" it by replacing it with its greatest common divisor with each subsequence entry. Here's one implementation.

```
def gcd(a,b):
        while a!=0: a,b = b%a,a
        return b

def gcd_list(ls):
        res = ls[0]
        for n in ls:
                res = gcd(res,n)
        return res

ls = map(int,raw_input().split())
print gcd_list(ls)
```

Note that there is also a very "pythonic" one-liner that can also be used to implement `gcd_list`, as follows. It uses the fact that $\gcd(n, 0) = n$ for all $n \neq 0$.

```
def gcd_list(ls):
        return reduce(gcd,ls,0)
```

8. (*Extra credit*). Write a program which can solve problem 1 of this assignment automatically (i.e. decrypt a passage of English text encrypted with a Caesar cipher). (I am happy to provide hints for how to proceed).

   Solution. There are a number of ways to approach this problem. Using the code from problem 1, it is easy to generate the 26 "candidates" for the plaintext; the hard part is determining which one is in English.

   Students in the class found several different approaches to this; you should browse the submissions to see what sort of things were done. I'll describe one method here.

   One easy-to-code approach is to use *frequency analysis.* The basic idea is that the 26 letters do not occur equally often in English: some are seen more often than others. On page 6 of the textbook you can find the frequency of each of the 26 letters. You can similarly form the list of frequencies of all 26 letters in a candidate text, and attempt to measure how closely these frequencies mirror those of English. A simple way to quantify them is to regard both frequency lists as vectors with 26 coordinates and taking their dot product (if you then divide by the magnitude of the two vectors, you will obtain the cosine of the angle between them, i.e. the *correlation coefficient* of the two frequency lists; since the magnitudes of the vectors are the same for all 26 candidates, we can dispense with this last step and just compare the dot products). If implemented, this approach is good enough to break the Caesar cipher in all of the test cases online.

   A slightly more theoretically-grounded approach (that is more robust for use in other applications) is to compute a "likelihood score" for each candidate text. This score is computed

as follows: if p_i denotes the probability that a randomly chosen letter from an English text is the ith letter of the alphabet (so p_0 is the probability of `a`, p_1 is the probability of `b`, and so forth), and the letters of a text correspond to indices $i_0, i_1, i_2, \cdots, i_{l-1}$, then the *likelihood* of this string is defined to be the product

$$p_{i_0} p_{i_1} \cdots p_{i_{l-1}}.$$

This product tells you the probability that you would obtain this particular sequence of letters if you were to choose the letters randomly from the "English letter distribution." It should be higher for English passages than non-English passages. Since this number will grow very small very quickly, it is better in practice to compute its logarithm, the *log-likelihood*.

$$\log p_{i_0} + \log p_{i_1} + \cdots + \log p_{i_{l-1}}$$

If you compute the log-likelihood of each of the 26 candidate decipherings, you will generally find that one of these "scores" beats the others by a long shot; this will be the English plaintext. Here is an implementation (note that this also uses the code written in problem 1, which I will not reproduce here). I use a structure called a `dict` to record the letter frequencies for later computation; you could just as easily use an array, and access it by first computing the index of a letter in English (from 0 to 25).

```
import math

# Omitted: code for the rotate_char and rotate functions (see problem 1)

# A dictionary of letter frequencies, copied from p. 6 of the textbook.
freq = {
        'A' : 0.0815, 'B' : 0.0144, 'C' : 0.0276, 'D' : 0.0379,
        'E' : 0.1311, 'F' : 0.0292, 'G' : 0.0199, 'H' : 0.0526,
        'I' : 0.0635, 'J' : 0.0013, 'K' : 0.0042, 'L' : 0.0339,
        'M' : 0.0254, 'N' : 0.0710, 'O' : 0.0800, 'P' : 0.0198,
        'Q' : 0.0012, 'R' : 0.0683, 'S' : 0.0610, 'T' : 0.1047,
        'U' : 0.0246, 'V' : 0.0092, 'W' : 0.0154, 'X' : 0.0017,
        'Y' : 0.0198, 'Z' : 0.0008
}

def freq_score(line):
        res = 0
        for char in line:
                char = char.upper()
                if char in freq:
                        res += math.log(freq[char])
        return res

def decrypt(line):
        # Initialially, best score is minus infinity
```

```
        # (to ensure that any score will appear better).
        best_score = float("-inf")
        # Try each of 26 rotations, and remember the best frequency score.
        for n in range(26):
                rot_line = rotate(line,n)
                score = freq_score(rot_line)
                if score > best_score:
                        best_score = score
                        best_rot = rot_line
        return best_rot

# I/O code for online submission
line = raw_input()
print decrypt(line)
```