Timothy Wong

CPE 321

Professor DeBruhl

Lab 1

**File name:** Caesar\_easy\_encrypted.txt

**Encryption Key:** 8

**Decrypted text:**

“Scrooge took his melancholy dinner in his usual melancholy tavern; and having read all the newspapers, and beguiled the rest of the evening with his banker's-book, went home to bed. He lived in chambers which had once belonged to his deceased partner. They were a gloomy suite of rooms, in a lowering pile of building up a yard, where it had so little business to be, that one could scarcely help fancying it must have run there when it was a young house, playing at hide-and-seek with other houses, and forgotten the way out again. It was old enough now, and dreary enough, for nobody lived in it but Scrooge, the other rooms being all let out as offices. The yard was so dark that even Scrooge, who knew its every stone, was fain to grope with his hands. The fog and frost so hung about the black old gateway of the house, that it seemed as if the Genius of the Weather sat in mournful meditation on the threshold.”

**Process:**

I used brute force to test out all 25 different possibilities for the key and scanned the results for the correct decrypted message.

**Code:**  
f = open(**"caesar\_easy\_encrypted.txt"**, **"r"**)  
  
count = [0] \* 26  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
  
**for** i **in** range (0,25):  
 A[i] = A[i].upper()  
  
  
s = f.read()  
  
**for** x **in** range(1,25):  
 trans = **""  
 for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = A[(A.index(c) + x) % 26]  
 **else**:  
 c = a[(a.index(c) + x) % 26]  
 trans += c  
 print (**'Key #%s'** % (x))  
 print (trans)

**File name:** Caesar\_easy\_2\_encrypted.txt

**Encryption Key:** 15

**Decrypted text:**

“One of the phenomena which had peculiarly attracted my attention was the structure of the human frame, and, indeed, any animal endued with life. Whence, I often asked myself, did the principle of life proceed? It was a bold question, and one which has ever been considered as a mystery; yet with how many things are we upon the brink of becoming acquainted, if cowardice or carelessness did not restrain our inquiries. I revolved these circumstances in my mind and determined thenceforth to apply myself more particularly to those branches of natural philosophy which relate to physiology. Unless I had been animated by an almost supernatural enthusiasm, my application to this study would have been irksome and almost intolerable. To examine the causes of life, we must first have recourse to death. I became acquainted with the science of anatomy, but this was not sufficient; I must also observe the natural decay and corruption of the human body. In my education my father had taken the greatest precautions that my mind should be impressed with no supernatural horrors. I do not ever remember to have trembled at a tale of superstition or to have feared the apparition of a spirit. Darkness had no effect upon my fancy, and a churchyard was to me merely the receptacle of bodies deprived of life, which, from being the seat of beauty and strength, had become food for the worm. Now I was led to examine the cause and progress of this decay and forced to spend days and nights in vaults and charnel-houses. My attention was fixed upon every object the most insupportable to the delicacy of the human feelings. I saw how the fine form of man was degraded and wasted; I beheld the corruption of death succeed to the blooming cheek of life; I saw how the worm inherited the wonders of the eye and brain. I paused, examining and analysing all the minutiae of causation, as exemplified in the change from life to death, and death to life, until from the midst of this darkness a sudden light broke in upon me — a light so brilliant and wondrous, yet so simple, that while I became dizzy with the immensity of the prospect which it illustrated, I was surprised that among so many men of genius who had directed their inquiries towards the same science, that I alone should be reserved to discover so astonishing a secret.”

**Process:**

I used brute force to test out all 25 different possibilities for the key and scanned the results for the correct decrypted message. I then manually inserted spaces into the message as the original message did not contain spaces.

**Code:**

f = open(**"caesar\_easy\_2\_encrypted.txt"**, **"r"**)  
  
count = [0] \* 26  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
  
**for** i **in** range (0,25):  
 A[i] = A[i].upper()  
  
  
s = f.read()  
  
**for** x **in** range(1,25):  
 trans = **""  
 for** c **in** s:**if** (c.isalpha()):  
 **if** c.isupper():  
 c = A[(A.index(c) + x) % 26]  
 **else**:  
 c = a[(a.index(c) + x) % 26]  
 trans += c  
 print (**'Key #%s'** % (x))  
 print (trans)  
  
**//**

**File name:** Caesar\_hard\_encrypt.txt

**Encryption Key:** 20

**Decrypted text:**

“But this is not a claim that Man should stay always youthful. Supposing that that famous Spaniard, landing upon Florida's coral strands, had found that mythical Fountain of Youth; what a calamity for mankind! A world without maturity of thought; without man's full-grown muscular ability to construct mighty buildings, railroads and ships; a world without authors, doctors, savants, musicians; nothing but Youth! I can think of but a solitary approval of such a condition; for such a horror as war would not,-could not occur; for a child is, naturally, a small bunch of sympathy. I know that boys will "scrap;" also that "spats" will occur amongst girls; but, at such a monstrosity as killings by bombing towns, sinking ships, or mass annihilation of marching troops, childhood would stand aghast. Not a tiny bird would fall; nor would any form of gun nor facility for manufacturing it, insult that almost Holy purity of youthful thought. Anybody who knows that wracking sorrow brought upon a child by a dying puppy or cat, knows that childhood can show us that our fighting, our policy of "a tooth for a tooth," is abominably wrong.”

**Process:**

I used brute force to test out all 25 different possibilities for the key and scanned the results for the correct decrypted message.

**Code:**

f = open(**"caesar\_hard\_encrypt.txt"**, **"r"**)  
  
count = [0] \* 26  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
  
**for** i **in** range (0,26):  
 A[i] = A[i].upper()  
  
  
s = f.read()  
  
**for** x **in** range(1,25):  
 trans = **""  
 for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = A[(A.index(c) + x) % 26]  
 **else**:  
 c = a[(a.index(c) + x) % 26]  
 trans += c  
 print (**'Key #%s'** % (x))  
 print (trans)

**//**

**File name:** Caesar\_hard\_2\_encrypt.txt

**Encryption Key:** 3

**Decrypted text:**

“As Gadsby sat thinking thus, his plan was rapidly growing; and, in a month, was actually starting to work. How? You'll know shortly; but first, you should know this John Gadsby; a man of "around fifty;" a family man, and known throughout Branton Hills for his high standard of honor and altruism on any kind of an occasion for public good. A loyal churchman, Gadsby was a man who, though admitting that an occasional fault in our daily acts is bound to occur, had taught his two boys and a pair of girls that, though folks do slip from what Scriptural authors call that "straight and narrow path," it will not pay to risk your own Soul by slipping, just so that you can laugh at your ability in staying out of prison; for Gadsby, having grown up in Branton Hills, could point to many such man or woman. So, with such firm convictions in his mind, this upstanding man was constantly striving so to act that no complaint from man, woman or child should bring a word of disapproval. In his mind, what a man might do was that man's affair only and could stain no Soul but his own. And his altruism taught that it is not difficult to find many ways in which to bring joy to such as cannot, through physical disability, go out to look for it; and that only a small bit of joy, brought to a shut-in will carry with it such a warmth as can flow only from acts of human sympathy.”

**Process:**

I used brute force to test out all 25 different possibilities for the key and scanned the results for the correct decrypted message. I then manually inserted spaces into the message as the original message did not contain spaces.

**Code:**

f = open(**"caesar\_hard\_2\_encrypt.txt"**, **"r"**)  
  
count = [0] \* 26  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
  
**for** i **in** range (0,26):  
 A[i] = A[i].upper()  
  
  
s = f.read()  
  
**for** x **in** range(1,25):  
 trans = **""  
 for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = A[(A.index(c) + x) % 26]  
 **else**:  
 c = a[(a.index(c) + x) % 26]  
 trans += c  
 print (**'Key #%s'** % (x))  
 print (trans)

**//**

**File name:** mono\_easy\_encrypt.txt

**Encryption Key:** wevyqsrxnltikgcjpdomf\*uhba

**Decrypted text:**

“Megalonyx (“great claw”) is the Greek name for another of the giant ground sloths. The name was proposed by Thomas Jefferson in 1797, based on fossil specimens found in a cave in West Virginia. Megalonyx jeffersonii, of the family Megalonychidae, was a large, heavily built animal about 8 to 10 feet (2.5–3 m) long. Its maximum weight may have been as much as 800 pounds. This is medium-sized among the giant ground sloths.

Like other ground sloths it had a blunt snout, massive jaw nd large, peg-like teeth. The hind limbs were plantigrade (flat-footed) and this, along with its stout tail, allowed it to rear up into a semi-erect position to feed on tree leaves. The forelimbs had three highly developed claws that were probably used to strip leaves and tear off branches.

M. jeffersonii was apparently the most wide-ranging giant ground sloth. Fossils are known from many Pleistocene sites in the United States, including most of the states east of the Rocky Mountains as well as along the west coast. It was the only ground sloth to range as far north as the present-day Yukon and Alaska.

In 2010, the first specimen ever found in Colorado was discovered at the Ziegler Reservoir site near Snowmass Village (in the Rocky Mountains at an elevation of 8,874 feet). Why the Giant Ground Sloth, as with other megafauna of the Miocene epoch, grew to such enormous size is a mystery.

Besides their bulk, these sloths were distinguished by their significantly longer front than hind legs, a clue that they used their long front claws to rope in copious amounts of vegetation. As big as it was, though, Megalonyx was a mere pup compared to the truly giant Megatherium.

Megatherium and Megalonyx are distant relatives of today’s modern Two- and Three-fingered sloths that live in Central and South America.

**Process:**

I created a list of objects that hold a letter of the alphabet and the number of times it appears in the text. I also created another list of the same type that holds the frequencies of letters appearing in the English language. I then manually created the mono-alphabetic by matching the most frequent letters in the encrypted text with the most frequent letters in the English language and using trial and error to find the letters that match.

**Code:**

**class** Letter():  
 **def** \_\_init\_\_(self, letter, f):  
 self.letter = letter  
 self.f = f  
  
f = open(**"mono\_easy\_encrypt.txt"**, **"r"**)  
  
freq = []  
expected = []  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
count = 0  
engl = [0.08176, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966, 0.00153, 0.00772,  
 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978,  
 0.02360, 0.00150, 0.01974, 0.00074]  
chi = 0  
  
**for** i **in** range (26):  
 A[i] = A[i].upper()  
 freq.append(Letter(a[i],0))  
 expected.append((Letter(a[i],0)))  
  
s = f.read()  
  
**for** c **in** s.lower():  
 **if** c.isalpha():  
 freq[(a.index(c))].f += 1  
 count += 1  
**for** i **in** range (26):  
 expected[i].f = engl[i] \* count  
  
freq.sort(key=**lambda** letter:letter.f, reverse=**True**)  
expected.sort(key=**lambda** letter:letter.f, reverse=**True**)  
  
*# for i in range (26):  
# print ('%s, %s' % (freq[i].letter, freq[i].f))  
# print('expected %s, %s' % (expected[i].letter, expected[i].f))*key = [**"\*"**] \* 26  
key[a.index(**"b"**)] = **"e"**key[a.index(**"k"**)] = **"t"**key[a.index(**"z"**)] = **"a"**key[a.index(**"f"**)] = **"s"**key[a.index(**"s"**)] = **"o"**key[a.index(**"i"**)] = **"n"**key[a.index(**"l"**)] = **"i"**key[a.index(**"g"**)] = **"r"**key[a.index(**"x"**)] = **"h"**key[a.index(**"j"**)] = **"l"**key[a.index(**"n"**)] = **"g"**key[a.index(**"r"**)] = **"d"**key[a.index(**"t"**)] = **"m"**key[a.index(**"w"**)] = **"u"**key[a.index(**"u"**)] = **"f"**key[a.index(**"a"**)] = **"w"**key[a.index(**"d"**)] = **"y"**key[a.index(**"o"**)] = **"c"**key[a.index(**"q"**)] = **"p"**key[a.index(**"c"**)] = **"v"**key[a.index(**"y"**)] = **"b"**key[a.index(**"m"**)] = **"k"**key[a.index(**"h"**)] = **"x"**key[a.index(**"p"**)] = **"j"**key[a.index(**"e"**)] = **"q"**trans = **""  
for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = key[A.index(c)]  
 **else**:  
 c = key[a.index(c)]  
 trans += c  
print (trans)

**//**

**File name:** mono\_medium\_encrypt.txt

**Encryption Key:** VIMEXBPHSTWGFJCL\*DYARNOZU\*

**Decrypted text:**

“TWAS BRILLIG, AND THE SLITHY TOVES

DID GYRE AND GIMBLE IN THE WABE:

ALL MIMSY WERE THE BOROGOVES,

AND THE MOME RATHS OUTGRABE.

BEWARE THE JABBERWOCK, MY SON!

THE JAWS THAT BITE, THE CLAWS THAT CATCH!

BEWARE THE JUBJUB BIRD, AND SHUN

THE FRUMIOUS BANDERSNATCH!

HE TOOK HIS VORPAL SWORD IN HAND;

LONG TIME THE MANXOME FOE HE SOUGHT

SO RESTED HE BY THE TUMTUM TREE

AND STOOD AWHILE IN THOUGHT.

AND, AS IN UFFISH THOUGHT HE STOOD,

THE JABBERWOCK, WITH EYES OF FLAME,

CAME WHIFFLING THROUGH THE TULGEY WOOD,

AND BURBLED AS IT CAME!

ONE, TWO! ONE, TWO! AND THROUGH AND THROUGH

THE VORPAL BLADE WENT SNICKER-SNACK!

HE LEFT IT DEAD, AND WITH ITS HEAD

HE WENT GALUMPHING BACK.

AND HAST THOU SLAIN THE JABBERWOCK?

COME TO MY ARMS, MY BEAMISH BOY!

O FRABJOUS DAY! CALLOOH! CALLAY!

HE CHORTLED IN HIS JOY.

TWAS BRILLIG, AND THE SLITHY TOVES

DID GYRE AND GIMBLE IN THE WABE:

ALL MIMSY WERE THE BOROGOVES,

AND THE MOME RATHS OUTGRABE.”

**Process:**

I created a list of objects that hold a letter of the alphabet and the number of times it appears in the text. I also created another list of the same type that holds the frequencies of letters appearing in the English language. I then manually created the mono-alphabetic by matching the most frequent letters in the encrypted text with the most frequent letters in the English language and using trial and error to find the letters that match.

**Code:**

**class** Letter():  
 **def** \_\_init\_\_(self, letter, f):  
 self.letter = letter  
 self.f = f  
  
f = open(**"mono\_medium\_encrypt.txt"**, **"r"**)  
  
freq = []  
expected = []  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
count = 0  
engl = [0.08176, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966, 0.00153, 0.00772,  
 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978,  
 0.02360, 0.00150, 0.01974, 0.00074]  
chi = 0  
  
**for** i **in** range (26):  
 A[i] = A[i].upper()  
 freq.append(Letter(a[i],0))  
 expected.append((Letter(a[i],0)))  
  
s = f.read()  
  
**for** c **in** s.lower():  
 **if** c.isalpha():  
 freq[(a.index(c))].f += 1  
 count += 1  
**for** i **in** range (26):  
 expected[i].f = engl[i] \* count  
  
freq.sort(key=**lambda** letter:letter.f, reverse=**True**)  
expected.sort(key=**lambda** letter:letter.f, reverse=**True**)  
  
**for** i **in** range (26):  
 print (**'%s, %s'** % (freq[i].letter, freq[i].f))  
 print(**'expected %s, %s'** % (expected[i].letter, expected[i].f))  
  
key = [**"\*"**] \* 26  
key[a.index(**"x"**)] = **"e"**key[a.index(**"a"**)] = **"t"**key[a.index(**"h"**)] = **"h"**key[a.index(**"v"**)] = **"a"**key[a.index(**"c"**)] = **"o"**key[a.index(**"y"**)] = **"s"**key[a.index(**"s"**)] = **"i"**key[a.index(**"j"**)] = **"n"**key[a.index(**"d"**)] = **"r"**key[a.index(**"e"**)] = **"d"**key[a.index(**"i"**)] = **"b"**key[a.index(**"g"**)] = **"l"**key[a.index(**"f"**)] = **"m"**key[a.index(**"o"**)] = **"w"**key[a.index(**"p"**)] = **"g"**key[a.index(**"r"**)] = **"u"**key[a.index(**"m"**)] = **"c"**key[a.index(**"u"**)] = **"y"**key[a.index(**"b"**)] = **"f"**key[a.index(**"t"**)] = **"j"**key[a.index(**"w"**)] = **"k"**key[a.index(**"n"**)] = **"v"**key[a.index(**"l"**)] = **"p"**key[a.index(**"z"**)] = **"x"**trans = **""  
for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = key[A.index(c)].upper()  
 **else**:  
 c = key[a.index(c)]  
 trans += c  
print (trans)

**File name:** vigerene\_easy\_encrypted.txt

**Encryption Key:** LEWIS

**Decrypted text:**

the room displayed a modest and pleasant color-scheme, after one of the best standard designs of the decorator who “did the interiors” for most of the speculative-builders’ houses in zenith. the walls were gray, the woodwork white, the rug a serene blue; and very much like mahogany was the furniture—the bureau with its great clear mirror, mrs. babbitt’s dressing-table with toilet-articles of almost solid silver, the plain twin beds, between them a small table holding a standard electric bedside lamp, a glass for water, and a standard bedside book with colored illustrations—what particular book it was cannot be ascertained, since no one had ever opened it. the mattresses were firm but not hard, triumphant modern mattresses which had cost a great deal of money; the hot-water radiator was of exactly the proper scientific surface for the cubic contents of the room. the windows were large and easily opened, with the best catches and cords, and holland roller-shades guaranteed not to crack. it was a masterpiece among bedrooms, right out of cheerful modern houses for medium incomes. only it had nothing to do with the babbitts, nor with any one else. if people had ever lived and loved here, read thrillers at midnight and lain in beautiful indolence on a sunday morning, there were no signs of it. it had the air of being a very good room in a very good hotel. one expected the chambermaid to come in and make it ready for people who would stay but one night, go without looking back, and never think of it again.

**Process:**

I created a list of objects that hold a letter of the alphabet and the frequency it appears in the text. I also created another list of the same type that holds the frequencies of letters appearing in the English language. Because I know that the key will be length 5, 9, or 13, I tried first with a key length of 5. I split the ciphertext into 5 groups with one containing every fifth letter after (including) the first letter and another containing every fifth letter after the second letter and so on. From there for each group, I multiplied the frequency of the letters from the group with the frequency of the letters in the English language (found online). I then shifted the letters from the group by one space with wrapping and then performed the same multiplication. The max product from each group became the letters for the key. With the key then, I subtracted each letter from the ciphertext with the key repeated in vigerene fashion to get the decrypted message.

**Code:**  
**class** Letter():  
 **def** \_\_init\_\_(self, letter, f):  
 self.letter = letter  
 self.f = f  
  
**def** shift(list, s):  
 **return** list[s:] + list[:s]  
  
**def** multAlpha(l1, l2):  
 tot = 0  
 **for** i **in** range(26):  
 tot += l1[i].f \* l2[i].f  
 **return** tot  
  
  
f = open(**"vigerene\_easy\_encrypted.txt"**, **"r"**)  
  
freq = []  
expected = []  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
count = 0  
engl = [0.08176, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966, 0.00153, 0.00772,  
 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978,  
 0.02360, 0.00150, 0.01974, 0.00074]  
n = 5  
  
**for** i **in** range (26):  
 A[i] = A[i].upper()  
 freq.append(Letter(a[i],0))  
 expected.append((Letter(a[i],engl[i])))  
  
s = f.read()  
  
**for** i **in** range (n):  
 count = 0  
 max = 0  
 spot = 0  
 tot = 0  
  
 temp = **""  
 for** c **in** s.lower():  
 **if** c.isalpha():  
 **if** (count - i) % n == 0:  
 freq[a.index(c)].f += 1  
 count += 1  
 **for** j **in** range (26):  
 freq[j].f = freq[j].f/ float(count)  
 **for** j **in** range (26):  
 tot = multAlpha(shift(freq,j),expected)  
 **if** max < tot:  
 max = tot  
 spot = j  
 print (**'max : %s, letter : %s, %s'** % (max, a[spot - 1], spot))  
  
key = [12,5,23,9,19]  
  
trans = **""**count = 0  
**for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = A[(A.index(c) + (26-key[count % n]))%26]  
 **else**:  
 c = a[(a.index(c) + (26-key[count % n]))%26]  
 count += 1  
 trans += c  
  
print (trans)

**File name:** vigerne\_medium\_encrypt.txt

**Encryption Key:** CACTACEAE

**Decrypted text:**

acanthaceae

achariaceae

achatocarpaceae

acoraceae

actinidiaceae

adoxaceae

aextoxicaceae

aizoaceae

akaniaceae

alismataceae

alseuosmiaceae

alstroemeriaceae

altingiaceae

alzateaceae

amaranthaceae

amaryllidaceae

amborellaceae

anacardiaceae

anarthriaceae

ancistrocladaceae

anisophylleaceae

annonaceae

aphanopetalaceae

aphloiaceae

apiaceae

apocynaceae

apodanthaceae

aponogetonaceae

aquifoliaceae

araceae

araliaceae

arecaceae

argophyllaceae

aristolochiaceae

asparagaceae

asteliaceae

asteropeiaceae

atherospermataceae

austrobaileyaceae

balanopaceae

balanophoraceae

balsaminaceae

barbeuiaceae

barbeyaceae

basellaceae

bataceae

begoniaceae

berberidaceae

berberidopsidaceae

betulaceae

biebersteiniaceae

bignoniaceae

bixaceae

blandfordiaceae

bonnetiaceae

boraginaceae

boryaceae

brassicaceae

bromeliaceae

brunelliaceae

bruniaceae

burmanniaceae

burseraceae

butomaceae

buxaceae

byblidaceae

cabombaceae

cactaceae

calceolariaceae

calophyllaceae

calycanthaceae

calyceraceae

campanulaceae

campyneumataceae

canellaceae

cannabaceae

cannaceae

capparaceae

caprifoliaceae

cardiopteridaceae

caricaceae

carlemanniaceae

caryocaraceae

caryophyllaceae

casuarinaceae

celastraceae

centrolepidaceae

centroplacaceae

cephalotaceae

ceratophyllaceae

cercidiphyllaceae

chloranthaceae

chrysobalanaceae

circaeasteraceae

cistaceae

cleomaceae

clethraceae

clusiaceae

colchicaceae

columelliaceae

combretaceae

commelinaceae

compositae

connaraceae

convolvulaceae

coriariaceae

cornaceae

corsiaceae

corynocarpaceae

costaceae

crassulaceae

crossosomataceae

crypteroniaceae

ctenolophonaceae

cucurbitaceae

cunoniaceae

curtisiaceae

cyclanthaceae

cymodoceaceae

cynomoriaceae

cyperaceae

cyrillaceae

cytinaceae

daphniphyllaceae

dasypogonaceae

datiscaceae

degeneriaceae

diapensiaceae

dichapetalaceae

didiereaceae

dilleniaceae

dioncophyllaceae

dioscoreaceae

dipentodontaceae

dipterocarpaceae

dirachmaceae

doryanthaceae

droseraceae

drosophyllaceae

ebenaceae

ecdeiocoleaceae

elaeagnaceae

elaeocarpaceae

elatinaceae

emblingiaceae

ericaceae

eriocaulaceae

erythroxylaceae

escalloniaceae

eucommiaceae

euphorbiaceae

euphroniaceae

eupomatiaceae

eupteleaceae

fagaceae

flagellariaceae

fouquieriaceae

frankeniaceae

garryaceae

geissolomataceae

gelsemiaceae

gentianaceae

geraniaceae

gesneriaceae

gisekiaceae

gomortegaceae

goodeniaceae

goupiaceae

griseliniaceae

grossulariaceae

grubbiaceae

gunneraceae

gyrostemonaceae

haemodoraceae

halophytaceae

haloragaceae

hamamelidaceae

hanguanaceae

haptanthaceae

heliconiaceae

helwingiaceae

hernandiaceae

himantandraceae

huaceae

humiriaceae

hydatellaceae

hydnoraceae

hydrangeaceae

hydrocharitaceae

hydroleaceae

hydrostachyaceae

hypericaceae

hypoxidaceae

icacinaceae

iridaceae

irvingiaceae

iteaceae

ixioliriaceae

ixonanthaceae

joinvilleaceae

juglandaceae

juncaceae

juncaginaceae

kirkiaceae

koeberliniaceae

krameriaceae

lacistemataceae

lactoridaceae

lamiaceae

lanariaceae

lardizabalaceae

lauraceae

lecythidaceae

leguminosae

lentibulariaceae

lepidobotryaceae

liliaceae

limeaceae

limnanthaceae

linaceae

linderniaceae

loasaceae

loganiaceae

lophiocarpaceae

loranthaceae

lowiaceae

lythraceae

magnoliaceae

malpighiaceae

malvaceae

marantaceae

marcgraviaceae

martyniaceae

mayacaceae

melanthiaceae

melastomataceae

meliaceae

melianthaceae

menispermaceae

menyanthaceae

metteniusaceae

misodendraceae

mitrastemonaceae

molluginaceae

monimiaceae

montiaceae

montiniaceae

moraceae

moringaceae

muntingiaceae

musaceae

myodocarpaceae

myricaceae

myristicaceae

myrothamnaceae

myrtaceae

nartheciaceae

nelumbonaceae

nepenthaceae

neuradaceae

nitrariaceae

nothofagaceae

nyctaginaceae

nymphaeaceae

ochnaceae

olacaceae

oleaceae

onagraceae

oncothecaceae

opiliaceae

orchidaceae

orobanchaceae

oxalidaceae

paeoniaceae

pandaceae

pandanaceae

papaveraceae

paracryphiaceae

passifloraceae

paulowniaceae

pedaliaceae

penaeaceae

pentadiplandraceae

pentaphragmataceae

pentaphylacaceae

penthoraceae

peraceae

peridiscaceae

petermanniaceae

petrosaviaceae

philesiaceae

philydraceae

phrymaceae

phyllanthaceae

phyllonomaceae

physenaceae

phytolaccaceae

picramniaceae

picrodendraceae

piperaceae

pittosporaceae

plantaginaceae

platanaceae

plocospermataceae

plumbaginaceae

poaceae

podostemaceae

polemoniaceae

polygalaceae

polygonaceae

pontederiaceae

portulacaceae

posidoniaceae

potamogetonaceae

primulaceae

proteaceae

putranjivaceae

quillajaceae

rafflesiaceae

ranunculaceae

rapateaceae

resedaceae

restionaceae

rhabdodendraceae

rhamnaceae

rhipogonaceae

rhizophoraceae

roridulaceae

rosaceae

rousseaceae

rubiaceae

ruppiaceae

rutaceae

sabiaceae

salicaceae

salvadoraceae

santalaceae

sapindaceae

sapotaceae

sarcobataceae

sarcolaenaceae

sarraceniaceae

saururaceae

saxifragaceae

scheuchzeriaceae

schisandraceae

schlegeliaceae

schoepfiaceae

scrophulariaceae

setchellanthaceae

simaroubaceae

simmondsiaceae

siparunaceae

sladeniaceae

smilacaceae

solanaceae

sphaerosepalaceae

sphenocleaceae

stachyuraceae

staphyleaceae

stegnospermataceae

stemonaceae

stemonuraceae

stilbaceae

strasburgeriaceae

strelitziaceae

stylidiaceae

styracaceae

surianaceae

symplocaceae

talinaceae

tamaricaceae

tapisciaceae

tecophilaeaceae

tetrachondraceae

tetramelaceae

tetrameristaceae

theaceae

thomandersiaceae

thurniaceae

thymelaeaceae

ticodendraceae

tofieldiaceae

torricelliaceae

tovariaceae

trigoniaceae

triuridaceae

trochodendraceae

tropaeolaceae

typhaceae

ulmaceae

urticaceae

vahliaceae

velloziaceae

verbenaceae

violaceae

vitaceae

vivianiaceae

vochysiaceae

winteraceae

xanthorrhoeaceae

xeronemataceae

xyridaceae

zingiberaceae

zosteraceae

zygophyllaceae

**Process:**

I created a list of objects that hold a letter of the alphabet and the frequency it appears in the text. I also created another list of the same type that holds the frequencies of letters appearing in the English language. Because I know that the key will be length 5, 9, or 13, I tried with a key length of 9. I split the ciphertext into 5 groups with one containing every fifth letter after (including) the first letter and another containing every fifth letter after the second letter and so on. From there for each group, I multiplied the frequency of the letters from the group with the frequency of the letters in the English language (found online). I then shifted the letters from the group by one space with wrapping and then performed the same multiplication. The max product from each group became the letters for the key. With the key then, I subtracted each letter from the ciphertext with the key repeated in vigerene fashion to get the decrypted message.

**Code:**

**class** Letter():  
 **def** \_\_init\_\_(self, letter, f):  
 self.letter = letter  
 self.f = f  
  
**def** shift(list, s):  
 **return** list[s:] + list[:s]  
  
**def** multAlpha(l1, l2):  
 tot = 0  
 **for** i **in** range(26):  
 tot += l1[i].f \* l2[i].f  
 **return** tot  
  
  
f = open(**"vigerne\_medium\_encrypt.txt"**, **"r"**)  
  
freq = []  
expected = []  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
count = 0  
engl = [0.08176, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966, 0.00153, 0.00772,  
 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978,  
 0.02360, 0.00150, 0.01974, 0.00074]  
n = 9  
  
**for** i **in** range (26):  
 A[i] = A[i].upper()  
 freq.append(Letter(a[i],0))  
 expected.append((Letter(a[i],engl[i])))  
  
s = f.read()  
  
**for** i **in** range (n):  
 count = 0  
 max = 0  
 spot = 0  
 tot = 0  
  
 temp = **""  
 for** c **in** s.lower():  
 **if** c.isalpha():  
 **if** (count - i) % n == 0:  
 freq[a.index(c)].f += 1  
 count += 1  
 **for** j **in** range (26):  
 freq[j].f = freq[j].f/ float(count)  
 **for** j **in** range (26):  
 tot = multAlpha(shift(freq,j),expected)  
 **if** max < tot:  
 max = tot  
 spot = j  
 print (**'max : %s, letter : %s, %s'** % (max, a[spot - 1], spot))  
  
key = [3,1,3,20,1,3,5,1,5]  
  
trans = **""**count = 0  
**for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = A[(A.index(c) + (26-key[count % n]))%26]  
 **else**:  
 c = a[(a.index(c) + (26-key[count % n]))%26]  
 count += 1  
 trans += c  
  
print (trans)

**//**

**File name:** vigerene\_hard\_encrypt.txt

**Encryption Key:** SNARKSAREREAL

**Decrypted text:**

the bellman’s speech

the bellman himself they all praised to the skies—

such a carriage, such ease and such grace!

such solemnity, too! one could see he was wise,

the moment one looked in his face!

he had bought a large map representing the sea,

without the least vestige of land:

and the crew were much pleased when they found it to be

a map they could all understand.

“what’s the good of mercator’s north poles and equators,

tropics, zones, and meridian lines?”

so the bell man would cry: and the crew would reply

“they are merely conventional signs!

“other maps are such shapes, with their is lands and capes!

but we’ve got our brave captain to thank:”

(so the crew would protest) “that he’s bought us the best—

a perfect and absolute blank!”

this was charming, no doubt; but they shortly found out

that the captain they trusted so well

had only one notion for crossing the ocean,

and that was to tingle his bell.

he was thoughtful and grave—but the order she gave

were enough to bewilder a crew.

when he cried “steer to starboard, but keep her head larboard!”

what on earth was the helmsman to do?

then the bow sprit got mixed with the rudder sometimes:

a thing, as the bellman remarked,

that frequently happens in tropical climes,

when a vessel is, so to speak, “snarked.”

but the principal failing occurred in the sailing,

and the bellman, perplexed and distressed,

said he had hoped, at least, when the wind blew due east,

that the ship would not travel due west!

but the danger was past—they had landed at last,

with their boxes, portmanteaus, and bags:

yet at first sight the crew were not pleased with the view,

which consisted of chasms and crags.

the bellman perceived that their spirits were low,

and repeated in musical tone

some jokes he had kept for a season of woe—

but the crew would do nothing but groan.

he served out some grog with a liberal hand,

and bade them sit down on the beach:

and they could not but own that their captain looked grand,

a she stood and delivered his speech.

“friends, romans, and countrymen, lend me your ears!”

(they were all of them fond of quotations:

so they drank to his health, and they gave him three cheers,

while he served out additional rations).

“we have sailed many months, we have sailed many weeks,

(four weeks to the month you may mark),

but never as yet(‘tis your captain who speaks)

have we caught the least glimpse of a snark!

“we have sailed many weeks, we have sailed many days,

(seven days to the week i allow),

but a snark, on the which we might lovingly gaze,

we have never beheld till now!

“come, listen, my men, while i tell you again

the five unmistakable marks

by which you may know, where so ever you go,

the warranted genuine snarks.

“let us take them in order. the first is the taste,

which is meagre and hollow, but crisp:

like a coat that is rather too tight in the waist,

with a flavour of will-o’-the-wisp.

“its habit of getting up late you’ll agree

that it carries too far, when i say

that it frequently breakfast sat five-o’ clock tea,

and dines on the following day.

“the third is its slowness in taking a jest.

should you happen to venture on one,

it will sigh like a thing that is deeply distressed:

and it always looks grave at a pun.

“the fourth is its fondness for bathing-machines,

which is constantly carries about,

and believes that they add to the beauty of scenes—

a sentiment open to doubt.

“the fifth is ambition. it next will be right

to describe each particular batch:

distinguishing those that have feathers, and bite,

and those that have whiskers, and scratch.

“for, although common snarks do no manner of harm,

yet, i feel it my duty to say,

some are boojums—”the bellman broke off in alarm,

for the baker had fainted away.

**Process:**

I created a list of objects that hold a letter of the alphabet and the frequency it appears in the text. I also created another list of the same type that holds the frequencies of letters appearing in the English language. Because I know that the key will be length 5, 9, or 13, I tried with a key length of 13. I split the ciphertext into 5 groups with one containing every fifth letter after (including) the first letter and another containing every fifth letter after the second letter and so on. From there for each group, I multiplied the frequency of the letters from the group with the frequency of the letters in the English language (found online). I then shifted the letters from the group by one space with wrapping and then performed the same multiplication. The max product from each group became the letters for the key. With the key then, I subtracted each letter from the ciphertext with the key repeated in vigerene fashion to get the decrypted message. I then added spaces to the deciphered text file.

**Code:**  
**class** Letter():  
 **def** \_\_init\_\_(self, letter, f):  
 self.letter = letter  
 self.f = f  
  
**def** shift(list, s):  
 **return** list[s:] + list[:s]  
  
**def** multAlpha(l1, l2):  
 tot = 0  
 **for** i **in** range(26):  
 tot += l1[i].f \* l2[i].f  
 **return** tot  
  
  
f = open(**"vigerene\_hard\_encrypt.txt"**, **"r"**)  
  
freq = []  
expected = []  
a = [**"a"**,**"b"**,**"c"**,**"d"**,**"e"**,**"f"**,**"g"**,**"h"**,**"i"**,**"j"**,**"k"**,**"l"**,**"m"**,**"n"**,**"o"**,**"p"**,**"q"**,**"r"**,**"s"**,**"t"**,**"u"**,**"v"**,**"w"**,**"x"**,**"y"**,**"z"**]  
A = a[:]  
count = 0  
engl = [0.08176, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966, 0.00153, 0.00772,  
 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978,  
 0.02360, 0.00150, 0.01974, 0.00074]  
n = 13  
  
**for** i **in** range (26):  
 A[i] = A[i].upper()  
 freq.append(Letter(a[i],0))  
 expected.append((Letter(a[i],engl[i])))  
  
s = f.read()  
  
**for** i **in** range (n):  
 count = 0  
 max = 0  
 spot = 0  
 tot = 0  
  
 temp = **""  
 for** c **in** s.lower():  
 **if** c.isalpha():  
 **if** (count - i) % n == 0:  
 freq[a.index(c)].f += 1  
 count += 1  
 **for** j **in** range (26):  
 freq[j].f = freq[j].f/ float(count)  
 **for** j **in** range (26):  
 tot = multAlpha(shift(freq,j),expected)  
 **if** max < tot:  
 max = tot  
 spot = j  
 print (**'max : %s, letter : %s, %s'** % (max, a[spot - 1], spot))  
  
key = [19,14,1,18,11,19,1,18,5,18,5,1,12]  
  
trans = **""**count = 0  
**for** c **in** s:  
 **if** (c.isalpha()):  
 **if** c.isupper():  
 c = A[(A.index(c) + (26-key[count % n]))%26]  
 **else**:  
 c = a[(a.index(c) + (26-key[count % n]))%26]  
 count += 1  
 trans += c  
  
print (trans)