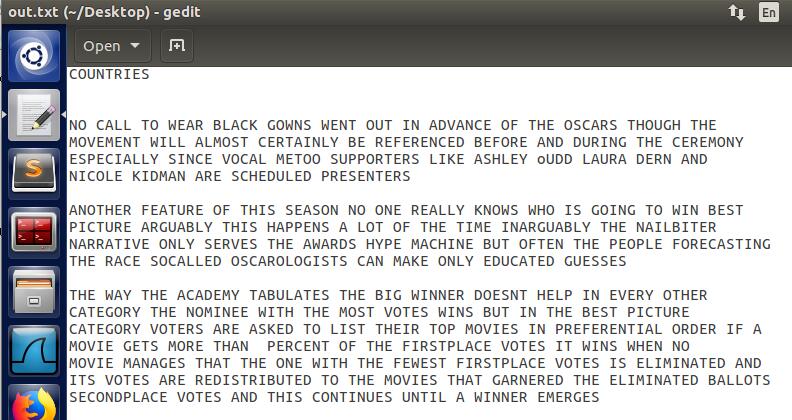
**Task 1**

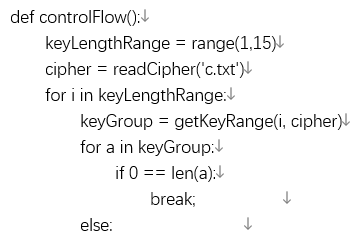
tr ‘nyvtxmuphzbqiadrlcgesfkj’ ‘ETAHOINDRUFSLCYGWMBPKVXQ’ <in.txt>out.txt



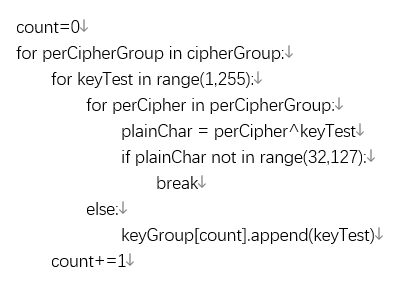
**Task 2**

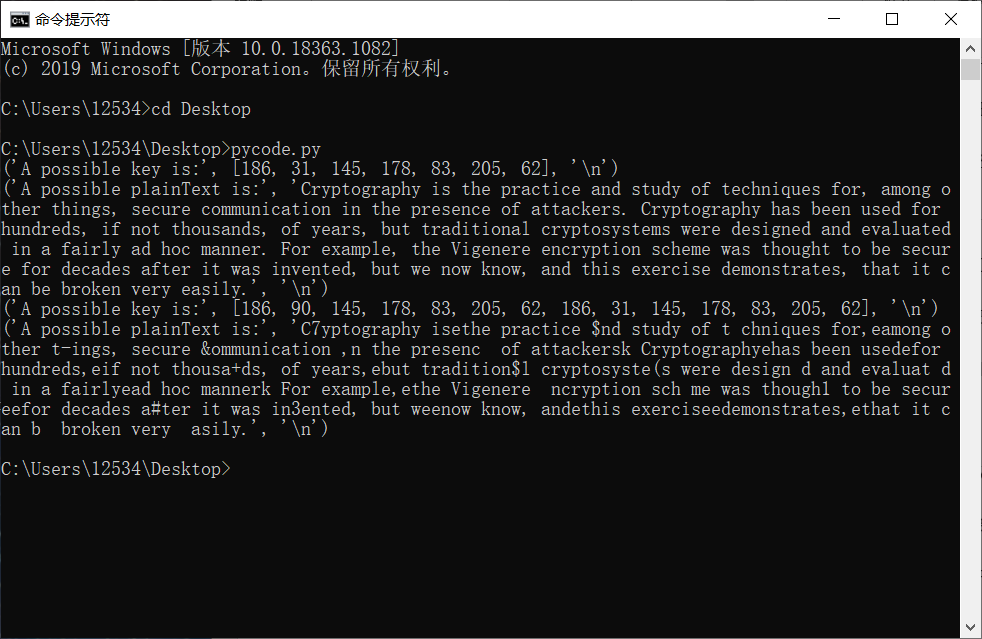
本题的密文是由明文与密钥异或而来的，因此 明文^密钥 = 密文，也就是明文 = 密文^密钥。关键点是，明文一定是由ascii字符组成的，并且密钥一定存在。如果假定出密钥的长度，那么可以将密文划分成相应的几组，密钥的取值范围又在一定的范围之内（unsigned char ，0~255）。因此，如果在假定的长度下，某一组密文在这个范围内找不到合理的密钥（使得与密文异或为ascii字符的密钥），那就说明密钥不可能是这个长度。

首先，求出1~14范围内密钥的取值，如果有空集就证明不能为这个长度：



通过如下确定密钥长度求出每一位密钥的可能值：



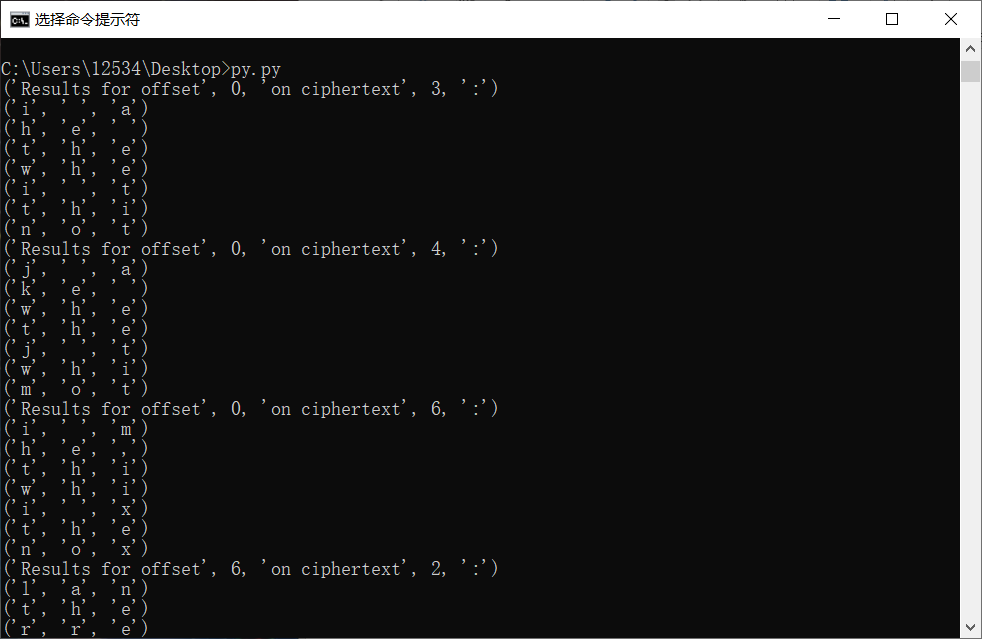


**完整代码：**

def readCipher(filename):  
 file = open(filename, 'r')  
 strCipher = file.read()  
 cipher = []  
 index = 0  
 while index < len(strCipher):  
 cipher.append(int(strCipher[index:index+2], 16))  
 index += 2  
 return cipher  
  
def getCipherGroup(keyLength, cipher):  
 cipherGroup = [[] for a in range(keyLength)]  
 count = 0  
 while count < len(cipher):  
 cipherGroup[(count) % keyLength] += [cipher[count]]  
 count += 1  
 return cipherGroup  
  
  
def getKeyRange(keyLength, cipher):  
 cipherGroup = getCipherGroup(keyLength, cipher)  
 keyGroup = [[] for a in range(keyLength)]  
  
 count=0  
 for perCipherGroup in cipherGroup:  
 for keyTest in range(1,255):  
 for perCipher in perCipherGroup:  
 plainChar = perCipher^keyTest  
 if plainChar not in range(32,127):  
 break  
 else:  
 keyGroup[count].append(keyTest)  
 count+=1  
  
 return keyGroup  
  
def getLetterFrequency(key, perCipher):  
 frequencies = {"e": 0.12702, "t": 0.09056, "a": 0.08167, "o": 0.07507, "i": 0.06966,  
 "n": 0.06749, "s": 0.06327, "h": 0.06094, "r": 0.05987, "d": 0.04253,  
 "l": 0.04025, "c": 0.02782, "u": 0.02758, "m": 0.02406, "w": 0.02360,  
 "f": 0.02228, "g": 0.02015, "y": 0.01974, "p": 0.01929, "b": 0.01492,  
 "v": 0.00978, "k": 0.00772, "j": 0.00153, "x": 0.00150, "q": 0.00095,  
 "z": 0.00074}  
  
 count={}  
 for ch in perCipher:  
 plainChar = key^ch  
 if plainChar in range(65,91) or plainChar in range(97,123):  
 char = chr(plainChar).lower()  
 count[char] = count.setdefault(char, 0)+1  
  
 freq = 0.0  
 for a in count:  
 freq += frequencies[a]\*count[a]/len(perCipher)  
  
 return freq  
  
def getKeyConfirmValue(keyGroup,cipher):  
 cipherGroup = getCipherGroup(len(keyGroup), cipher)  
 key = []  
  
 count=0  
 for perKey in keyGroup:  
 maxFreq = 0  
 tempKey = 0  
 for a in perKey:  
 freq = getLetterFrequency(a, cipherGroup[count])  
 if freq>maxFreq:  
 maxFreq = freq  
 tempKey = a  
 key.append(tempKey)  
 count += 1  
   
 return key  
  
def cipherDecrypt(key, cipher):  
 plainText = ''  
 index = 0  
 for a in cipher:  
 plainText += chr(key[index%len(key)]^a)  
 index += 1  
 return plainText  
  
def controlFlow():  
 keyLengthRange = range(1,15)  
 cipher = readCipher('c.txt')  
 for i in keyLengthRange:  
 keyGroup = getKeyRange(i, cipher)  
 for a in keyGroup:  
 if 0 == len(a):  
 break;   
 else:   
 #print('A possible key group is:', keyGroup,'\n')  
 key = getKeyConfirmValue(keyGroup, cipher)  
 print('A possible key is:', key,'\n')  
 plainText = cipherDecrypt(key, cipher)  
 print('A possible plainText is:', plainText,'\n')  
  
if \_\_name\_\_=='\_\_main\_\_':  
 controlFlow()

**Task 3**

def verify\_results(results):  
    for i in results:  
        if i < 32 or i > 122:   
            return False  
        else:  
            if i < 48:  
                if i != 32 and i != 33 and i != 44 and i != 45 and i != 46: return False  
            elif i > 57 and i < 65:  
                if i != 63 and i != 58 and i != 59: return False  
            elif i > 90 and i < 97:  
                return False  
    return True   
  
# List of ciphertexts encrypted with the same key  
c = ["BB3A65F6F0034FA957F6A767699CE7FABA855AFB4F2B520AEAD612944A801E",  
     "BA7F24F2A35357A05CB8A16762C5A6AAAC924AE6447F0608A3D11388569A1E",  
     "A67261BBB30651BA5CF6BA297ED0E7B4E9894AA95E300247F0C0028F409A1E",  
     "A57261F5F0004BA74CF4AA2979D9A6B7AC854DA95E305203EC8515954C9D0F",  
     "BB3A70F3B91D48E84DF0AB702ECFEEB5BC8C5DA94C301E0BECD241954C831E",  
     "A6726DE8F01A50E849EDBC6C7C9CF2B2A88E19FD423E0647ECCB04DD4C9D1E",  
     "BC7570BBBF1D46E85AF9AA6C7A9CEFA9E9825CFD5E3A0047F7CD009305A71E"]  
  
# Put ciphertexts into byte arrays  
for i,e in enumerate(c):  
    c[i] = bytearray.fromhex(e)  
  
# ASCII representation of the word, "the"  
the\_ascii = [116, 104, 101]  
  
# Try for each offset  
for i in range(0, 28):  
      
# Try for each ciphertext  
    for n in range(0, 7):  
        crib\_word = [the\_ascii[0]^c[n][i], the\_ascii[1]^c[n][i+1], the\_ascii[2]^c[n][i+2]]         
        ciphers = []  
        legit = True  
          
        for j in range(0, 7):  
            results = [crib\_word[0]^c[j][i], crib\_word[1]^c[j][i+1], crib\_word[2]^c[j][i+2]]  
              
            if verify\_results(results):  
                ciphers.append(results)  
            else:  
                # If the verification fails, end  
                legit = False  
                break  
                  
        if legit:  
            print ("Results for offset", i, "on ciphertext", n+1, ":")  
            for x in ciphers:  
                print (chr(x[0]), chr(x[1]), chr(x[2]) )



观察结果，有3个结果没有特殊的符号，对于英语句子有意义：

('Results for offset', 0, 'on ciphertext', 3, ':')

('i', ' ', 'a')

('h', 'e', ' ')

('t', 'h', 'e')

('w', 'h', 'e')

('i', ' ', 't')

('t', 'h', 'i')

('n', 'o', 't')

Results for offset 6 on ciphertext 2 :

l a n

t h e

r r e

h o u

k t

s p

e c

Results for offset 8 on ciphertext 5 :

n n i

e o

e n t

u l d

t h e

p u r

c a d

Results for offset 19 on ciphertext 7 :

r y t

o r

h o

h o

z o

t t a

t h e

经过推测发现第四组数据不符合英语习惯。

再依据than->should->person->to->follow->one->mission可以猜出全部密文。

I\_am\_planning\_a\_secret\_mission.

He\_is\_the\_only\_person\_to\_trust.

The\_current\_plan\_is\_top\_secret.

When\_should\_we\_meet\_to\_do\_this?

I\_think\_they\_should\_follow\_him.

This\_is\_purer\_than\_that\_one\_is.

Not\_one\_cadet\_is\_better\_than\_1.