Github 账号：Joel-Q-Xu

实验题目：破解重复键 XOR

实验摘要：

1. 实验网站：http://cryptopals.com/sets/1/challenges/6
   利用 python 编写解密代码程序。
2. 已使用重复秘钥的异或对明文文本进行加密，并进行 base64 编码，得到此文件，试解密得到明文文本。
3. 可利用前面 5 题 1 将 hex 转换为 base64，2 固定异或，3.单字符 XOR 密码，4 检测单字符异或和 5 实现重复键 XOR 来实现此实验内容。

题目描述

题目

已使用重复秘钥的异或对明文文本进行加密，并对其进行 base64 编码，得到此文件，试解密得到明文文本。

提示：

1.首先猜测密钥长度，记为 keysize，建议猜测范围为 2 到 40。

2.写一个计算两个字符串的 Hamming distance（汉明距离）的函数。Hamming distance 是两个字符串不同比特的数量。例如，this is a test 和 wokka wokka!!!之间的 Hamming distance 为 37。

3.对于每一个 keysize，获得密文文本的第一个长度为 keysize 的块以及第二个长度为 keysize 的块，计算这两个块的 Hamming distance 并除以 keysize 得到结果记为 avg_distance。

4.有着最小的 avg_distance 的 keysize 很可能就是密钥的长度。可以选择最小的三个 avg_distance 所对应的 keysize 进行接下来的操作。也可以在第 3 步中使用 4 个 keysize 大小的块来计算得到 avg_distance。

5.将密文分为长度为 keysize 的块。

6.将每个块中的第一个字节组合成一个新的块，每个块中的第二个字节组合成一个新的块，依次类推。

7.使用单字节 XOR 密码来处理每个新块。即第一个新块中的每个字节均和密钥的第一个字节异或，第二个新块中的每个字节均和密钥的第二个字节异或，依次类推。由此可得到每个 keysize 所对应的候选密钥 key。

实验设想：

读取和解码文件内容，通过循环潜在的密钥大小范围，编写函数来计算汉明距离，规范汉明距离，将密文分解为块确定的密钥长度和转置块，然后实现 XOR 暴力破解。

过程

1、 汉明距离 Hamming distance

我们以 d（x,y）表示两个字 x,y 之间的汉明距离。对两个字符串进行异或运算，并统计结果为 1 的个数，那么这个数就是汉明距离。——百度百科

因此代码编写如下：

```
#以字节型传入
def hamming_distance(m1, m2):
    """我们以d（x,y）表示两个字x,y之间的汉明距离。对两个字符串进行异或运算，
       并统计结果为1的个数，那么这个数就是汉明距离。——百度百科
    """
    hamming_distance = 0
    for b1, b2 in zip(m1,m2):
        x = b1^b2
        hamming_distance += sum([1 for bit in bin(x) if bit == '1'])
    return hamming_distance
```

尝试题目中的例子 this is a test 和 wokka wokka!!!之间的 Hamming distance 为 37，检测函数是否正确。

输入代码：

```
def main() :
    m1='this is a test'
    mm1=bytearray.fromhex((m1.encode()).hex())
    m2='wokka wokka!!!'
    mm2=bytearray.fromhex((m2.encode()).hex())

    print(hamming_distance(mm1,mm2))

if __name__ == '__main__':
    main()
```

运行，得出结果：

```
========== RESTART: D:\python\cryptopals\set1\hamming_distance.py ==========
37
```

则函数编写正确。

2、 单字符 XOR 加密

编写单个字符的 XOR 加密函数：

```
def single_xor(ciphertext,i):
    #返回使用单个值进行异或的每个字节的结果
    plaintext = b''   #令plaintext为bytes型
    for item in ciphertext:
        plaintext += bytes([item ^i])
    return plaintext
```

3、 利用频率分析来破解密码

编写频率字典并编写得分代码：

```
def get_score(m):
    """将每个输入字节与字符频率表进行比较，并根
据English语言中字符出现的相对频率返回消息的分数
    """
    freq = {
        "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, ' ': 0.13000, "\n":0
    }
    score=0.0
    for i in m.lower():
        score+=freq.get(chr(i),0)
    return score
```

4、 破解单字符 XOR

编写函数，利用单字符加密函数 single_xor()和得分函数 get_score()来破解单字符异或，代码如下，返回得分最高即最有可能的密文：

```
def break_single_xor(ciphertext):
    """对每个可能的值（0,255）执行单字符异或，
    并根据字符频率分配分数。 返回得分最高的结果。
    """
    messages = []
    for i in range(256):
        m = single_xor(ciphertext,i)
        score = get_score(m)
        data = {
            'message': m,
            'score': score,
            'key':chr(i)
            }
        messages.append(data)
    #排序(升序),得到最高分，输出sorted[-1]
    best_m=sorted(messages, key=lambda x: x['score'])[-1]
    return best_m
```

5、 实现重复键 XOR，代码如下

```
def xor(m,key):
    #如果消息长于密钥，则密钥将重复。
    c = b''   #令plaintext为bytes型
    i=0
    for item in m:
        c+=bytes([item^key[i]])
        if (i+1)==len(key):
            i=0
        else:
            i+=1
    return c
```

6、 破解重复秘钥 XOR

1) 设置一个空列表 avg_distance[]以存放两个块的 Hamming distance 并除以 keysize 的结果

2) 由于猜测密钥长度 keysize 范围为 2 到 40

3) 设置一个空列表 distances[]以存放两个块的 Hamming distance

4) 将密文分成密钥长度的块

以上操作代码如下：

```
avg_distance = []
# 钥匙的长度：尝试从2到（比如说）40的值
for keysize in range(2,41):
    #初始化列表以存储此密钥大小的汉明距离
    distances = []
    # 将密文分成密钥长度的块
    blocks = [ciphertext[i:i+keysize] for i in range(0, len(ciphertext), keysize)]
```

5) 对于每一个 keysize，获得密文文本的第一个长度为 keysize 的块以及第二个长度为 keysize 的块,，计算这两个块的 Hamming distance 并插入列表 distances[]

然后删除这些块，以便在循环重新开始时,可以计算下个 keysize 中两个块的汉明距离

以上代码操作如下：

```
# 从列表开头的两个块中获取汉明距离,除以KEYSIZE将此结果标准化
block1 = blocks[0]
block2 = blocks[1]
distance = hamming_distance(block1, block2)
distances.append(distance/keysize)

#删除这些块，以便在循环重新开始时,可以计算下两个块的汉明距离
del blocks[0]
del blocks[1]
```

6) 建立字典，包含元素 keysize 和 avg_dist，由于 distances 每组都不同，消除不同 distances 的影响，突出字母频率，避免个数的影响，

即计算 avg_dist=sum(distances)/len(distances)，

将其插入列表 avg_distance[]。

7) 提示中指出有着最小的 avg_distance 的 keysize 很可能就是密钥的长度，由于由于 distances 每组都不同，为消除不同 distances 的影响，突出字母频率，避免个数的影响，此时最小的 avg_dist 就可能是密钥的长度。由于不同 distances 针对不同 keysize，计算最小值时已跳出 keysize 的 for 循环。

以上代码如下：

```
    avg= {
        'key': keysize,
        'avg_dist': sum(distances)/len(distances)
        }
    avg_distance.append(avg)
#升序，即avg_distancr[0]最小
possible_keysize = sorted(avg_distance,key=lambda x: x['avg_dist'])[0]
```

8) 设置一个空列表 possible_plaintext = []待存放破解出明文与密钥，

设置变量 key 为一个空 bytes

令 possible_length=最小的 avg_dist 所对应的列表 avg_distance[]中元素中的键‘key’对应的值 keysize，即 possible_length 就是密钥最大可能的长度。

9) 将每个块中的第一个字节组合成一个新的块，每个块中的第二个字节组合成一个新的块，依次类推。此时，每个块都使用同一个单字符加密。

每个新块为以下代码中的 block：

```
for i in range(possible_length):
    block = b''
    for j in range(i, len(ciphertext), possible_length):
        block += bytes([ciphertext[j]])
```

10) 使用单字节 XOR 密码来处理每个新块，即第一个新块中的每个字节均和密钥的第一个字节异或，第二个新块中的每个字节均和密钥的第二个字节异或，依次类推。求出每个块对应的单字符密钥，由此可得到该最大可能 keysize 所对应的候选密钥 key。

利用挑战 4 破解单字符中的 break_single_xor()函数即可得到每个新块对应的最有可能的明文，密钥和对应得分。将 key 转换为 ascii 码并放入 bytes 数组中

```
key += bytes([ord(break_single_xor(block)['key'])])
```

11) 因为 k^m=c,所以 m=k^c，我们将所得到 key 与密文再次异或，利用实验 5 中重复密钥 XOR 的 xor（）函数即可得到明文，将明文和对应密钥放入

```
possible_plaintext.append((xor(ciphertext, key), key))
```

12)

7、 编写主函数

读取文件，将其进行 base64 解密

利用上述 6 中函数 break_xor()得到明文与密文，代码如下：

```
def main():
    c=open('c6.txt').read()
    ciphertext=base64.b64decode(c)
    m, key = break_xor(ciphertext)
    print(key.decode(),'\n',m.decode())


if __name__ == '__main__':
    main()
```

输出如下：

```
keysize: 29
key:Terminator X: Bring the noise
m:I'm back and I'm ringin' the bell
A rockin' on the mike while the fly girls yell
In ecstasy in the back of me
Well that's my DJ Deshay cuttin' all them Z's
Hittin' hard and the girlies goin' crazy
Vanilla's on the mike, man I'm not lazy.
```

密文未截取完整

总代码如下：

```
import base64
def get_score(m):
    """将每个输入字节与字符频率表进行比较，并根
据 Englis 语言中字符出现的相对频率返回消息的分数
    """
    freq = {
        "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, ' ': 0.13000, "\n":0
    }
    score=0.0
    for i in m.lower():
        score+=freq.get(chr(i),0)
```

```
    return score


def single_xor(ciphertext,i):
    #返回使用单个值进行异或的每个字节的结果
    plaintext = b''  #令 plaintext 为 bytes 型
    for item in ciphertext:
        plaintext += bytes([item ^i])
    return plaintext


def break_single_xor(ciphertext):
    """对每个可能的值（0,255）执行单字符异或，
    并根据字符频率分配分数。 返回得分最高的结果。
    """
    messages = []
    for i in range(256):
        m = single_xor(ciphertext,i)
        score = get_score(m)
        data = {
            'message': m,
            'score': score,
            'key':chr(i)
            }
        messages.append(data)
    #排序(升序),得到最高分，输出 sorted[-1]
    best_m=sorted(messages, key=lambda x: x['score'])[-1]
    return best_m


def xor(m,key):
    #如果消息长于密钥，则密钥将重复。
    c = b''  #令 plaintext 为 bytes 型
    i=0
    for item in m:
        c+=bytes([item^key[i]])
        if (i+1)==len(key):
```

```
            i=0
        else:
            i+=1
    return c

#以字节型传入
def hamming_distance(m1, m2):
    """我们以 d（x,y）表示两个字 x,y 之间的汉明距离。对两个字符串进行异或运算，
        并统计结果为 1 的个数，那么这个数就是汉明距离。——百度百科
    """
    hamming_distance = 0
    for b1, b2 in zip(m1,m2):
        x = b1^b2
        hamming_distance += sum([1 for bit in bin(x) if bit == '1'])
    return hamming_distance

def break_xor(ciphertext):
    avg_distance = []
    # 钥匙的长度：尝试从 2 到（比如说）40 的值
    for keysize in range(2,41):
        #初始化列表以存储此密钥大小的汉明距离
        distances = []
        # 将密文分成密钥长度的块
        blocks = [ciphertext[i:i+keysize] for i in range(0, len(ciphertext), keysize)]
        while True:
            try:
                # 从列表开头的两个块中获取汉明距离,除以 KEYSIZE 将此结果标准化
                block1 = blocks[0]
                block2 = blocks[1]
                distance = hamming_distance(block1, block2)
                distances.append(distance/keysize)

                #删除这些块，以便在循环重新开始时,可以计算下两个块的汉明距离
                del blocks[0]
```

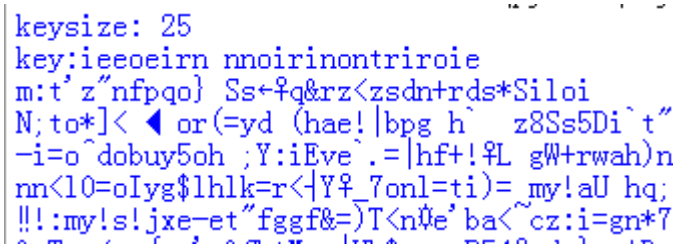
```
                del blocks[1]

            # 当发生异常（已处理所有块）时，跳出循环
            except Exception :
                break
        avg= {
            'key': keysize,
            'avg_dist': sum(distances)/len(distances)
            }
        avg_distance.append(avg)
    #升序，即 avg_distancr[0]最小
    possible_keysize = sorted(avg_distance,key=lambda x: x['avg_dist'])[0]

    possible_plaintext = []
    key = b''
    possible_length = possible_keysize['key']
    print('keysize:',possible_length)
    for i in range(possible_length):
        block = b''
        for j in range(i, len(ciphertext), possible_length):
            block += bytes([ciphertext[j]])
        key += bytes([ord(break_single_xor(block)['key'])])
    possible_plaintext.append((xor(ciphertext, key), key))
    return possible_plaintext[0]

def main():
    c=open('c6.txt').read()
    ciphertext=base64.b64decode(c)
    m,key= break_xor(ciphertext)
    print('key:{0}\nm:{1}'.format(key.decode(),m.decode()))

if __name__ == '__main__':
    main()
```

总结

1、 由于先前题目分开做，所使用函数输入输出格式各不相同，统一至实验六时发生大量报错，经修改，统一规定为字节型输入输出。

2、 由于提示中表述为对于每一个 keysize，获得密文文本的第一个长度为 keysize 的块以及第二个长度为 keysize 的块，计算这两个块的 Hamming distance 并除以 keysize 得到结果记为 avg_distance。 有着最小的 avg_distance 的 keysize 很可能就是密钥的长度。

因此直接将 distances 中元素 distance/keysize 进行排序，结果为：



后考虑到由于 distances 每组都不同，我们应消除不同 distances 的影响，突出字母频率，避免个数的影响，因此设置 avg_dist=sum(distances)/len(distances)，将其进行排序，结果正确

3、可在算出 keysize=29 后直接暴力破解密文使得出明文均为合法字符

参考文献

汉明距离：https://trustedsignal.blogspot.com/2015/06/xord-play-normalized-hamming-distance.html

https://github.com/nasume/cryptopals/blob/master/Set1/break_repeating_key_XOR.py