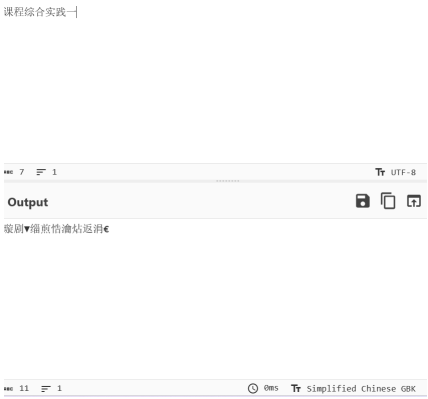
Task1

1.1 复现乱码情况

1.用 GBK 解码 UTF-8 编码的文本



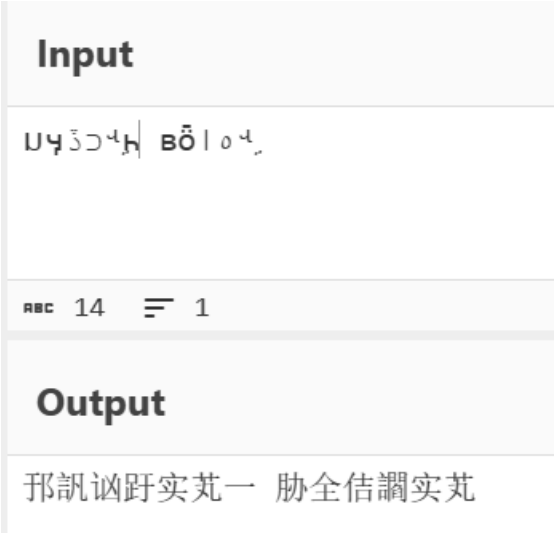
2.用 UTF-8 解码 GBK 编码的文本

3.用 latin-1 解码 UTF-8 编码的文本

4.用 latin-1 解码 GBK 编码的文本

5.先用 GBK 解码 UTF-8 编码的文本，再用 UTF-8 解码前面的结果

6.先用 UTF-8 解码 GBK 编码的文本，再用 GBK 解码前面的结果



1.2.1 GB 系列是如何实现三个版本兼容

- GB2312是最早的中文字符编码标准，后来GBK在GB2312的基础上进行了扩展，而GB18030是在GBK的基础上进一步扩展，因此GB2312可以被正确解析为GBK编码，GBK编码的文本也可被GB18030编码解码
- 由于GB18030是最新的中文字符编码标准，它包含了GB2312和GBK中所有字符，因此GB18030编码的文本可被GB2312和GBK编码解码。
- GB2312 GBK GB18030都采用多字节编码结构（使用不同的字节组合来表示不同的字符），这些编码标准在编码字符时使用了相似的编码规则因此它们之间也可以进行字符编码的相互转换，即可以实现三个版本兼容。

1.2.2 编解码的可恢复性

- 用utf-8解码gbk编码的文本不可恢复
- 用gbk解码utf-8编码的文本可恢复
- 用latin-1解码utf-8编码的文本可恢复
- 用latin-1解码gbk编码的文本可恢复

1.2.3 乱码“锟斤拷”的产生

- 锟斤拷乱码是由于以GBK方式读取UTF-8编码的��（方块码）得到
- Unicode字符集有一个专门用于提示用户字符无法识别或展示的替换符号
- 如果有UTF-8无法识别的字符，便会用这个问号替换。一个在UTF-8的十六进制表示为“EF BF BD"如果有两个连着的问号替换符，十六进制则为"EF BF BD EF BF BD"

- 而在GBK编码中，每个汉字使用两个字节
    - EF BF 对应汉字"锟"
    - BD EF对应汉字"斤"
    - BF BD对应汉字"拷"

因此使用GBK来读取UTF-8编码的��时，"��"就会被GBK读取成了"锟斤拷"

1.3 CyberChef 处理 UTF-8 解码错误的方法

可以通过如下方法

- 转化为十六进制 (from utf-8 to hex ,from hex to utf-8)
- 使用内置的utf-8 repair工具

Challenge 1

首先尝试获取密钥

爆破猜测密钥长度

- 针对每种可能的密钥长度 $k \in [15, 30]$,计算密文中第 i位和第$i+k$位的字符重合次数，选取重合次数最多的k即为最可能的密钥长度
- 代码实现：

```
def getKeyLength(cipher):
    keylength = 15
    max = 0
    for l in range(15,31):
        cnt = 0
        for i in range(len(cipher)-l):
            if cipher[i] == cipher[i+l]:
                cnt += 1
        if cnt>max:
            max = cnt
            keylength = l
    return keylength
```

可得密钥长度为29

逐位爆破密钥

- 确定密钥长度29后，密文中$i, i+k, i+2k$位的字符都是由同一个字母加密的，乘法加密不会改变字符的统计分布，可以通过26个字母的频率分布来猜测该位密钥
- 首先编写按长度29进行分组与计算其中字母频率分布并找出最大概率字母的函数

```
def texttolist(text,length):
```

```
    textMatrix = []
    row = []
    index = 0
    for ch in text:
        row.append(ch)
        index += 1
        if index % length == 0:
            textMatrix.append(row)
            row = []
    return textMatrix
def countpro(list):
    li = []
    alphabet = [chr(i) for i in range(97,123)]
    for c in alphabet:
        cnt = 0
        for ch in list:
            if ch == c:
                cnt += 1
        li.append(cnt/len(list))
    return li
def max(list):
    max = list[0]
    for x in list:
        if x > max:
            max = x
    return max
```

- 然后通过每个以单个字母加密的列表与字母e之间的关系，进行分析这一组的加密密钥

```
alpha =[]
for i in range(29):
    w = [row[i].lower() for row in matrix]
    li = countpro(w)
    max1 = max(li)
    cnt1 = li.index(max1)
    origindex = w[cnt1]
    textindex = text_list.index(origindex)
    print(origindex)
    alpha.append(textindex)

for num in alpha:
    for k in range(1,1000):
        if (69 * k) % 97 == num:
            k %= 97
            key.append(k)
            break
```

- 最后编写解密函数

```
def decrypt(c, k):
    out = ''
```

```
    for i in range(len(c)):
        index = text_list.index(c[i])
        index *= modular_inverse(k[i % len(k)], 97)
        index %= 97
        out += text_list[index]
    return out

def modular_inverse(a, m):
    if a==0:
        return 0
    for x in range(1, m):
        if (a * x) % m == 1:
            return x

cipher = open('cipher.txt','r').read()
decrypted = decrypt(cipher,key)
open('plain1.txt','w').write(decrypted)
```

做到这里本以为大功告成，但是解出来的文章仍然不太正常，思考并查阅资料发现并不是所有都满足频率分布规则，每组最多只有134个字符，统计规律可能不适用，我便将文章按如上的以长为29的单元分组，其中基本上全是英文字母的块符合e出现最多的规律，剩余的我想不到代码实现方法，故进行手动模拟。

最终得到密钥 `key = [51,14,22,90,67,43,17,82,48,17,41,33,89,94,60,80,13,72,79,18,65,50,20,6,36,39,56,7,65]`

代入上述代码得最终原文

```
1  By the mid-nineteenth century, the term "icebox" had entered the American language, but ice was still only beginning
2  Making an efficient icebox was not as easy as we might now suppose. In the early nineteenth century, the knowledge of
3  Nevertheless, early efforts to economize ice included wrapping the ice in blankets, which kept the ice from doing its
4  fLaG:AAA{i_like_T0ef1_v3ry_M3uh!!!}
5  But as early as 1803, an ingenious Maryland farmer, Thomas Moore, had been on the right track. He owned a farm about
6  One advantage of his icebox, Moore explained, was that farmers would no longer have to travel to market at night in o
7  Perhaps the most obvious way artistic creation reflects how people live is by mirroring the environment - the materia
8  What is particularly meaningful to anthropologist is the realization that although the materials available to a socie
```

得到flag如上图第4行所示

完整代码如下：

```
from random import randrange

text_list = ' !"#$%&\'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~\t\n'

key =
[51,14,22,90,67,43,17,82,48,17,41,33,89,94,60,80,13,72,79,18,65,50,20,6,36,39,56,7,6
5]

def decrypt(c, k):
    out = ''
    for i in range(len(c)):
        index = text_list.index(c[i])
        index *= modular_inverse(k[i % len(k)], 97)
        index %= 97
```

```
            out += text_list[index]
        return out

def modular_inverse(a, m):
    if a==0:
        return 0
    for x in range(1, m):
        if (a * x) % m == 1:
            return x

cipher = open('cipher.txt','r').read()
decrypted = decrypt(cipher,key)
open('plain1.txt','w').write(decrypted)
```

Challenge 2

2.1

使用exiftool可得

```
Create Date                     : 2023:02:09 21:28:31.122
Date/Time Original              : 2023:02:09 21:28:31.122+09:00
Modify Date                     : 2023:02:09 21:28:31.122+09:00
Thumbnail Image                 : (Binary data 14058 bytes, use -b option to extract)
GPS Altitude                    : 63.4 m Above Sea Level
GPS Date/Time                   : 2023:02:09 12:28:33Z
GPS Latitude                    : 35 deg 42' 11.67" N
GPS Longitude                   : 139 deg 45' 8.62" E
Focal Length                    : 7.6 mm
GPS Position                    : 35 deg 42' 11.67" N, 139 deg 45' 8.62" E
```

故所在位置高度为：63.4米

拍摄时间为：2023年2月9日 21点28分31秒

2.2

使用exiftool可得

```
Resolution Unit                 : inches
Modify Date                     : 2022:02:16 16:32:29
```

拍摄的月份：2月

当天的时间：16点32分

使用百度识图可识别该地区为厦门（海上明珠塔）

之后转到谷歌地图寻找具体位置，由拍照时间（下午四点）以及太阳方向，大致定位拍照地点在海上明珠塔南侧，由地图可大致确定该范围为白鹭洲公园北侧一带。但是因为寻找街景的时候我看3D图片头很晕，看卫星图看了很久也没有找对，故没有完成经纬度的精确定位。