Crypto寒假学习计划

第一周（01.15 — 01.21）

古典密码及Python实现

第二周（01.22 — 01.28）

一. 数学基本知识

二. Cryptopal中的"set1"和"set2"

第三周（01.29 — 02.04）

一. RSA算法的加解密原理及Python实现

二. Cryptopal中的"set3"和"set4"

第四周（02.05 — 02.11）

一. 刷crypto类ctf题目（在NSS上）

二. Cryptopal中的set5、set6、set7

第一周

古典密码及Python实现

一. Caesar crypto

原理：

通过将明文根据后移位数**(即密钥，范围为1~25)**进行加密，如下表所示（部分）：

原文	A/a	X/x	Y/y	Z/z
密文（key=3）	d	a	b	c

e.g.

明文：**cipher**

密钥：**3**

密文：**flskhu**

Python实现：

因为在计算机上，不可能直接进行字母与字母之间的转换；所以需要先将这些字母转换成对应的ASCII码，然后再进行转换。

于是需要用到ord()将字母转换成对应的ASCII码,如**ord(a)=97**

然后用判断语句进行大小写判断（设定密文均为小写字母）以及加密判断（存在一些后移位不够的情况，如key=3时的“z”应转化成“c"),

如

```
if(ord(x)-ord('a')+int(wheel)>26):
        print(chr(ord(x)+int(wheel)-26), end='')
```

最后分别把加密部分和解密部分编写出来，就完成了：

```
from numpy import place


def caesar_encryption(x):
    cipher=''
    plain = input("请输入明文:\n")
    plain= str(plain)
    wheel=input("请输入密钥:\n")
    for x in plain:
        if(x==' '):
            continue
        elif('a'<=x<='z'):
            if(ord(x)-ord('a')+int(wheel)>26):
                cipher+=chr(ord(x)+int(wheel)-26)
            else:
                cipher+=chr(ord(x)+int(wheel))
        elif('A'<=x<='Z'):
            if(ord(x)-ord('A')+int(wheel)>26):
                cipher+=chr(ord(x)+int(wheel)+6)
            else:
                cipher+=chr(ord(x)+int(wheel)+32)
        else:
            cipher+=x
    print("密文为:\n"+cipher)

def caesar_decryption(x):
    plain=''
    cipher=input("请输入密文:\n")
    cipher=str(cipher)
    wheel=input("请输入密钥:\n")
    for x in cipher:
        if('a'<=x<='z'):
            if(ord(x)-ord('a')-int(wheel)<0):
                plain+=chr(ord(x)-int(wheel)+26)
            else:
```

```
                plain+=chr(ord(x)-int(wheel))
        else:
            plain+=x
    print("明文为:\n"+plain)

if __name__=='__main__':
    x=input("加密请按1，解密请按2:\n")
    while(x!='1' and x!='2'):
        x=input("输入错误，加密请按1，解密请按2:\n")

    if(x=='1'):
        caesar_encryption(x)
    else:
        caesar_decryption(x)
```

二. Playfair crypto

原理:

通过密钥构成5X5的矩阵，然后对密文进行分组并一组一组地进行加密。**(其中字母j当作字母i来处理)**

加密规则:

1. 如果相邻两个字母相同，则在其中间插入一个字母(假设是字母q)；

2. 如果明文的字母总数为奇数，则在末尾添上一个字母(假设是字母q)。

3. 如果明文m1和m2在矩阵中的同一行，则密文c1和c2分别为紧靠m1和m2的右端字母（将第一列视为最后一列的右边）

4. 如果明文m1和m2在矩阵的同一列，则密文c1和c2分别为紧靠m1和m2的下方字母（将第一行视为最后一行的下方）

5. 如果明文m1和m2不符合3和4的情况，则密文c1和c2分别由m1和m2确定的矩形其他两个角的字母，c1和m1同行，c2和m2同行

e.g.

明文：**charles wheatstone**

密钥：**key**

1. 构造矩阵：

k e y a b

c d f g h

i l m n o

p q r s t

u v w x z

2.明文分组:

charles wheatstone

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓

charleswheatstoneq

3.加密:

明文	**ch**	**ar**	**le**	**sw**	**he**	**at**	**st**	**on**	**eq**
密文	**dc**	**sy**	**qd**	**rx**	**db**	**bs**	**tp**	**io**	**dv**

所以密文为**dcsyqdrxdbbstpiodv**

Python实现:

因为这种密码需要根据密钥来创造矩阵,所以需要编写一个函数来实现;于是根据网上关于矩阵的构建方法,编写了这个:

```
def Create_Cbook(key):
    key=Remove_Duplicates(key)  #移除密钥中的重复字母
    key=key.replace(' ','')     #去除密钥中的空格
    for ch in letter_list:      #根据密钥获取新组合的字母表
        if ch not in key:
            key+=ch
    j=0
    for i in range(0,len(key)): #将新的字母表里的字母逐个填入密码表中，组成5*5的矩阵
         Cbook[j]+=key[i]        #j用来定位字母表的行
         if (i+1)%5==0:
            j+=1
```

又因为这种加密方式有三种情况,且与字母矩阵中的位置有关;于是需要一个函数来获取字母的坐标,于是编写了这个:

```
def Get_Cbook(ch):
  for i in range(len(Cbook)):
    for j in range(len(Cbook)):
      if ch==Cbook[i][j]:
        return i,j #i为行，j为列
```

接着是加密部分,

第一种情况是两明文字母同行,则对应的密文字母分别为矩阵中明文字母右段的字母;因此只要将明文字母的横坐标加一，然后对其与五相除求余即可 。

同理，第二种情况是明文m1和m2在矩阵的同一列，则密文c1和c2分别为紧靠m1和m2的下方字母；只要将明文字母的纵坐标加一，然后对其与五求余即可 。

第三种情况是明文m1和m2不符合3和4的情况，则密文c1和c2分别由m1和m2确定的矩形其他两个角的字母，c1和m1同行，c2和m2同行；因此只要前者的纵坐标和后者的横坐标互换即可。

```
if x[0]==y[0]:    #如果在同一行
  cipher+=Cbook[x[0]][(x[1]+1)%5]
+Cbook[y[0]][(y[1]+1)%5]
elif x[1]==y[1]:  #如果在同一列
  cipher+=Cbook[(x[0]+1)%5][x[1]]
+Cbook[(y[0]+1)%5][y[1]]
else:             #如果不同行不同列
  cipher+=Cbook[x[0]][y[1]]+Cbook[y[0]][x[1]]
```

至于解密部分，其实是将上述代码反过来就行：

```
if x[0]==y[0]:    #如果在同一行
    plain+=Cbook[x[0]][(x[1]-1)%5]
+Cbook[y[0]][(y[1]-1)%5]
elif x[1]==y[1]:  #如果在同一列
    plain+=Cbook[(x[0]-1)%5][x[1]]
+Cbook[(y[0]-1)%5][y[1]]
else:             #如果不同行不同列
    plain+=Cbook[x[0]][y[1]]+Cbook[y[0]][x[1]]
```

最后整合在一起，就完成了：

```
#字母表
letter_list='abcdefghiklmnopqrstuvwxyz'

#密码表
Cbook=['','','','','']

#根据密钥建立密码表
def Create_Cbook(key):
    key=Remove_Duplicates(key)  #移除密钥中的重复字母
    key=key.replace(' ','')     #去除密钥中的空格
    for ch in letter_list:      #根据密钥获取新组合的字母表
        if ch not in key:
            key+=ch
    j=0
    for i in range(0,len(key)): #将新的字母表里的字母逐个填入密码表中，组成5*5的矩阵
         Cbook[j]+=key[i]        #j用来定位字母表的行
         if (i+1)%5==0:
            j+=1

#移除字符串中重复的字母
def Remove_Duplicates(key):
  key=key.lower() #转成小写字母组成的字符串
  _key=''
```

```
  for ch in key:
    if ch.lower()=='j':
      ch='i'
    if ch in _key:
      continue
    else:
      _key+=ch
  return _key

#获取字符在密码表中的位置
def Get_Cbook(ch):
  for i in range(len(Cbook)):
    for j in range(len(Cbook)):
      if ch==Cbook[i][j]:
        return i,j #i为行，j为列

#加密
def Encrypt(plain,Cbook):
    cipher=''
    plain=list(plain)
    i=0
    while i<len(plain): #对明文进行遍历
        if plain[i+1]==plain[i]:
            plain.insert(i+1,'q')
        if len(plain)%2 !=0:  #如果新的明文长度为奇数，在其末尾添上'q'
            plain+='q'
        if True==plain[i].isalpha():       #如果是明文是字母的话，
            j=0                            #则开始对该字母及其后一个字母进行遍历，
            while j<len(plain):            #然后遍历字母并进行加密
                j=i+1
                if(j==len(plain)):
                    break
                if True==plain[j].isalpha():
                    if plain[i].lower()=='j':
                        x=Get_Cbook('i')
                    else:
                        x=Get_Cbook(plain[i].lower()) #对字符在密码表中的坐标

                    if plain[j].lower()=='j':              #/进行定位,同时将'j'作
为'i'来处理
                        y=Get_Cbook('i')
                    else:
                        y=Get_Cbook(plain[j].lower())

                    if x[0]==y[0]:    #如果在同一行
                        cipher+=Cbook[x[0]][(x[1]+1)%5]+Cbook[y[0]][(y[1]+1)%5]
                    elif x[1]==y[1]:  #如果在同一列
                        cipher+=Cbook[(x[0]+1)%5][x[1]]+Cbook[(y[0]+1)%5][y[1]]
                    else:             #如果不同行不同列
                        cipher+=Cbook[x[0]][y[1]]+Cbook[y[0]][x[1]]
                j+=1
                i=j  #每次对明文的遍历是从加密过后的明文的后一个明文开始的
                continue
        else:
```

```
            cipher+=plain[i]  #如果明文不是字母，直接加到密文上
            i+=1
    return cipher

#解密
def Decrypt(cipher,Cbook):
    plain=''
    i=0
    while i<len(cipher): #对密文进行遍历
        if True==cipher[i].isalpha():    #如果是密文是字母的话，
            j=0                         #则开始对该字母及其后一个字母进行遍历，
            while j<len(cipher):          #然后遍历字母并进行解密
                j=i+1
                if True==cipher[j].isalpha():
                    if 'j'==cipher[i].lower():
                        x=Get_Cbook('i')
                    else:
                        x=Get_Cbook(cipher[i].lower())  #对字符在密码表中的坐标
                    if 'j'==cipher[j].lower():              #进行定位,同时将'j'作为
                        y=Get_Cbook('i')                  #'i'来处理
                    else:
                        y=Get_Cbook(cipher[j].lower())
                    if x[0]==y[0]:    #如果在同一行
                        plain+=Cbook[x[0]][(x[1]-1)%5]+Cbook[y[0]][(y[1]-1)%5]
                    elif x[1]==y[1]:  #如果在同一列
                        plain+=Cbook[(x[0]-1)%5][x[1]]+Cbook[(y[0]-1)%5][y[1]]
                    else:             #如果不同行不同列
                        plain+=Cbook[x[0]][y[1]]+Cbook[y[0]][x[1]]
                j+=1
                i=j  #每次对密文的遍历是从解密过后的密文的后一个密文开始的
                continue
        else:  #如果密文不是字母，直接加到明文上
            plain+=cipher[i]
            i+=1
    return plain


#主函数
if __name__=='__main__':
    user=input("加密请按1,解密请按2:\n")
    while(user!='1' and user!='2'):
        user=input("输入有误!请重新输入:\n")

    key=input("请输入你的密钥:\n")
    Create_Cbook(key)

    if user=='1':
        plain =input("请输入你的明文:\n")
        plain=plain.replace(' ','')
        print("密文为:\n"+Encrypt(plain,Cbook))
    else:
        cipher =input("请输入你的密文:\n")
        print("明文为:\n"+Decrypt(cipher,Cbook))
```

三. 矩形密码

原理：

将明文平均分成key行，然后一列一列地读取就得到密文了(如果明文内容数为奇数个时，在末尾加上x)

e.g.

明文：**transposition cipher**

密钥(key)：5

平均分成5行：

t r a n

s p o s

i t i o

n c i p

h e r x

于是密文为**tsinhrptceaoiirnsopx**

Python实现：

因为这是以分行的形式进行加密，所以我们可以让明文中的内容与每行的的个数进行求余来实现加密：

```
f = len(s) // n
for i in range(f):
        for j in range(0, len(s)):
            if j % f == i:
                c += s[j]
```

也因此，解密部分与之相似，只不过变成与行数进行求余：

```
for i in range(n):
        for j in range(0, len(s)):
            if j % n == i:
                c += s[j]
```

最后便得到整个加解密脚本：

```
def Rec_enc(s, n):
    s=str(s).lower()
    s=s.replace(" ","")
    for i in range(len(s)):
        if len(s)%2!=0:
            s=''.join([s,'x'])
    f = len(s) // n
    if(len(s)-n<n):
        print("输入无效，无法加密")
        return 0
    if(len(s)%n!=0):
        f+=1
    c = ''
    for i in range(f):
        for j in range(0, len(s)):
            if j % f == i:
                c += s[j]
    print("密文为:\n"+c)
    return c

def Rec_dec(s, n):
    c = ''
    for i in range(n):
        for j in range(0, len(s)):
            if j % n == i:
                c += s[j]
    return c

if __name__=='__main__':
    x=input("加密请按1，解密请按2\n")
    while(x!='1' and x!='2'):
        x=input("输入有误!请重新输入:\n")

    if x=='1':
        text=input("请输入明文:\n")
        key= eval(input('请输入密钥(行数):'))
        Rec_enc(text, key)
    else:
        text = input("请输入密文:\n")
        key= eval(input('请输入密钥(行数):'))
        print("明文为:\n"+Rec_dec(text, key))
```

四. 栅栏密码

原理:

将明文根据密钥摆成“W”型，然后逐行读取就得到密文了

e.g

明文：**railfence**

密钥：**2**

r i f n e

a l e c

最后得到密文：**rifnealec**

Python实现;

因为课本上关于栅栏密码的加密规律我没看懂，于是查了查网上关于栅栏密码的讲述。以下为收获：

根据上面的例子，可以发现:如果将密文中的“r”到“i”看作是一部分的话，则这个V字的长度为**3**，刚好是**2*2-1**；假如key=3，则这部分的长度为5，即**2*3-1**，如下列所示：

r f e

a l e c

i n

加密部分：

我们可以先在输出密文的地方分好key个组，然后让明文里的前 key个内容分别加到这key个组中，然后再反着回到中间的组中；循环往复，然后用一个判断语句来跳出循环，就完成了加密。加密部分如下所示：

```
def encode(string, key):#需要加密的字符串以及加密栏数
    i = 0
    enlist = []
    for j in range(0, key):
        enlist.append('')#添加分组，列表的一个元素相当于一个分组

    while i < len(string):#分组重排进行加密
        for k in range(0, key):
            if i >= len(string):
                break
            enlist[k] += string[i]
            i += 1
        for k in range(1, key-1):
            if i >= len(string):
                break
            enlist[key-1-k] += string[i]
            i += 1
        enstr = ''
    for i in range(key):
        enstr += enlist[i]
    return enstr
```

解密部分：

与加密过程类似，先通过密钥将密文分成key组；接着构建出包含key组的明文框架，然后进行来回填充，最后得到了明文。解密部分如下所示：

```
def decode(string, key):                #解密字符串以及解密栏数
    de_key = 2*key - 2                  #一个部分的长度
    length = len(string)//de_key        #确定有多少个完整部分
    r = len(string)%de_key              #最后不完整部分的长度
    delist = []
    for i in range(key):
        delist.append('')               #重新排布分组
    #确定第一个分组
    if r == 0:
        delist[0] += string[0:length]
        s = length
    else:
        delist[0] += string[0:length+1]
        s = length+1
    #确定第二个到第key-1个分组
    for i in range(1, key-1):
        l = length*2                    #这几个分组长度至少是完整部分数量的两倍
        #最后一个不完整部分对应当前分组有几个元素
        if r > i:
            l += 1
        if r > de_key-i:
            l += 1

        delist[i] += string[s:s+l]
        s = s+l
    #确定最后一个分组
    delist[key-1] += string[s:]
    #排布分组确定原文字符串
    destr = ''
    j = 0
    for i in range(0, len(string)):
        destr += delist[j][0]
        delist[j] = delist[j][1:]
        if j == key-1:
            flag = 0
        if j == 0:
            flag = 1
        if flag:
            j += 1
        else:
            j -= 1
    return destr
```

最后便得到了一个加解密脚本：

```
def encode(string, key):#需要加密的字符串以及加密栏数
    i = 0
    enlist = []
    for j in range(0, key):
        enlist.append('')#添加分组，列表的一个元素相当于一个分组
```

```
    while i < len(string):#分组重排进行加密
        for k in range(0, key):
            if i >= len(string):
                break
            enlist[k] += string[i]
            i += 1
        for k in range(1, key-1):
            if i >= len(string):
                break
            enlist[key-1-k] += string[i]
            i += 1
        enstr = ''
    for i in range(key):
        enstr += enlist[i]
    return enstr

def decode(string, key):              #解密字符串以及解密栏数
    de_key = 2*key - 2                #一个部分的长度
    length = len(string)//de_key      #确定有多少个完整部分
    r = len(string)%de_key            #最后不完整部分的长度
    delist = []
    for i in range(key):
        delist.append('')             #重新排布分组
    #确定第一个分组
    if r == 0:
        delist[0] += string[0:length]
        s = length
    else:
        delist[0] += string[0:length+1]
        s = length+1
    #确定第二个到第key-1个分组
    for i in range(1, key-1):
        l = length*2                  #这几个分组长度至少是完整部分数量的两倍
        #最后一个不完整部分对应当前分组有几个元素
        if r > i:
            l += 1
        if r > de_key-i:
            l += 1

        delist[i] += string[s:s+l]
        s = s+l
    #确定最后一个分组
    delist[key-1] += string[s:]
    #排布分组确定原文字符串
    destr = ''
    j = 0
    for i in range(0, len(string)):
        destr += delist[j][0]
        delist[j] = delist[j][1:]
        if j == key-1:
            flag = 0
        if j == 0:
            flag = 1
```

```
            if flag:
                j += 1
            else:
                j -= 1
        return destr


if __name__=='__main__':
    x=input("加密请按1，解密请按2\n")
    while(x!='1' and x!='2'):
        x=input("输入有误!请重新输入:\n")

    if(x=='1'):
        string=input("请输入明文:\n")
        key= eval(input('请输入栅栏深度(行数):'))
        print("密文为:\n"+encode(string,key))
    else:
        string = input("请输入密文:\n")
        key= eval(input('请输入栅栏深度(行数):'))
        print("明文为:\n"+decode(string, key))
```

五. 转轮机密码

原理：

通过使用转子来实现加密（其实质上是复杂的单表替代密码，即复式替代密码）

Python实现：

因为其实质是多次使用单表替换密码，所以需要设三个转子：

```
r_password1 = 'qwertyuiopasdfghjklzxcvbnm'
    r_password2 = 'asdfqwerzxcvtyuiopghjklbnm'
    r_password3 = 'poiuytrewqasdfghjklmnbvcxz'
```

而转轮机的特点就是"转"，即第一个转子转完一轮，第二个转子要转一格；第二个转完一轮，第三个要转一格。

因此需要一个函数及两条判断语句来实现这个功能：

```
if count % 676 == 0:    # 如果模676为0，那么转子3转动一次(因为转子2已经转动了一整圈)
            replace_word3 = rotors(replace_word3)
elif count % 26 == 0:  # 如果模26为0，那么转子2转动一次（因为转子1已经转动了一整圈)
            replace_word2 = rotors(replace_word2)

def rotors(replace_word):  # 转子转动的函数，每调用一次，就把转子前面第一个字母移动到最
后
    return replace_word[1:] + replace_word[0]
```

不仅如此，转轮机的奇特之处，在于反射器；有了这个反射器，便可实现“输入明文，得到密文；输入密文，得到明文“。因此需要一个函数来实现：

```
def reverse_word(word):
    dic = {'a': 'n', 'b': 'o', 'c': 'p', 'd': 'q',
           'e': 'r', 'f': 's', 'g': 't', 'h': 'u',
           'i': 'v', 'j': 'w', 'k': 'x', 'l': 'y',
           'm': 'z', 'n': 'a', 'o': 'b', 'p': 'c',
           'q': 'd', 'r': 'e', 's': 'f', 't': 'g',
           'u': 'h', 'v': 'i', 'w': 'j', 'x': 'k',
           'y': 'l', 'z': 'm'}
    return dic[word]
```

根据其原理，可以得到一个明密间转换的函数：

```
def simple_replace(password, replace_word1, replace_word2, replace_word3):  # 加密
的主函数
    count = 0  # 设置计数器
    new_pass = ''  # 设置一个空字符串准备接收密码
    ori_table = 'abcdefghijklmnopqrstuvwxyz'  # 原始的字符串，用来建立映射表
    for obj in password:  # 开始拆解原字符串
        table1 = str.maketrans(ori_table, replace_word1)  # 建立转子1的映射表
        table2 = str.maketrans(ori_table, replace_word2)  # 建立转子2的映射表
        table3 = str.maketrans(ori_table, replace_word3)  # 建立转子3的映射表
        new_obj = str.translate(obj, table1)  # 把obj通过转子1转换
        new_obj = str.translate(new_obj, table2)  # obj通过转子2
        new_obj = str.translate(new_obj, table3)  # obj通过转子3
        new_obj = reverse_word(new_obj)  # 进入自反器，得到自反值
        reverse_table1 = str.maketrans(replace_word1, ori_table)  # 增加自反出去的
对应表，反向解译
        reverse_table2 = str.maketrans(replace_word2, ori_table)
        reverse_table3 = str.maketrans(replace_word3, ori_table)
        new_obj = str.translate(new_obj, reverse_table3)  # new_obj再赋值，反向解译
通过转子3
        new_obj = str.translate(new_obj, reverse_table2)  # 通过转子2
        new_obj = str.translate(new_obj, reverse_table1)  # 通过转子1
        new_pass += new_obj  # 返回的密码增加一个new_obj
        replace_word1 = rotors(replace_word1)  # 转子1每个字符都转动一次
        count += 1  # 计数器增加1
        if count % 676 == 0:  # 如果模676为0，那么转子3转动一次(因为转子2已经转动了一
整圈)
            replace_word3 = rotors(replace_word3)
        elif count % 26 == 0:  # 如果模26为0，那么转子2转动一次（因为转子1已经转动了一
整圈)
            replace_word2 = rotors(replace_word2)
    return new_pass  # 返回新的已经被转子加密的密码
```

最终整合到一块，便得到了加解密脚本：

```
import re
import string


def simple_replace(password, replace_word1, replace_word2, replace_word3):  # 加密
的主函数
    count = 0  # 设置计数器
    new_pass = ''  # 设置一个空字符串准备接收密码
    ori_table = 'abcdefghijklmnopqrstuvwxyz'  # 原始的字符串，用来建立映射表
    for obj in password:  # 开始拆解原字符串
        table1 = str.maketrans(ori_table, replace_word1)  # 建立转子1的映射表
        table2 = str.maketrans(ori_table, replace_word2)  # 建立转子2的映射表
        table3 = str.maketrans(ori_table, replace_word3)  # 建立转子3的映射表
        new_obj = str.translate(obj, table1)  # 把obj通过转子1转换
        new_obj = str.translate(new_obj, table2)  # obj通过转子2
        new_obj = str.translate(new_obj, table3)  # obj通过转子3
        new_obj = reverse_word(new_obj)  # 进入自反器，得到自反值
        reverse_table1 = str.maketrans(replace_word1, ori_table)  # 增加自反出去的
对应表，反向解译
        reverse_table2 = str.maketrans(replace_word2, ori_table)
        reverse_table3 = str.maketrans(replace_word3, ori_table)
        new_obj = str.translate(new_obj, reverse_table3)  # new_obj再赋值，反向解译
通过转子3
        new_obj = str.translate(new_obj, reverse_table2)  # 通过转子2
        new_obj = str.translate(new_obj, reverse_table1)  # 通过转子1
        new_pass += new_obj  # 返回的密码增加一个new_obj
        replace_word1 = rotors(replace_word1)  # 转子1每个字符都转动一次
        count += 1  # 计数器增加1
        if count % 676 == 0:  # 如果模676为0，那么转子3转动一次(因为转子2已经转动了一
整圈)
            replace_word3 = rotors(replace_word3)
        elif count % 26 == 0:  # 如果模26为0，那么转子2转动一次（因为转子1已经转动了一
整圈)
            replace_word2 = rotors(replace_word2)
    return new_pass  # 返回新的已经被转子加密的密码




def is_str(password, replace_word):  # 判断的函数
    an = re.match('[a-z]+$', password)
    if not type(password) == type(replace_word) == type('a'):
        print('密码必须是字符串!')
        return False
    elif not an:
        print('字符串只能包含小写字母!')
        return False
    elif len(replace_word) != 26:
        print('替换码必须为26个字母!')
        return False
    else:
        return True
```

```
def rotors(replace_word):  # 转子转动的函数，每调用一次，就把转子前面第一个字母移动到最后
    return replace_word[1:] + replace_word[0]


# 反射器
def reverse_word(word):
    dic = {'a': 'n', 'b': 'o', 'c': 'p', 'd': 'q',
           'e': 'r', 'f': 's', 'g': 't', 'h': 'u',
           'i': 'v', 'j': 'w', 'k': 'x', 'l': 'y',
           'm': 'z', 'n': 'a', 'o': 'b', 'p': 'c',
           'q': 'd', 'r': 'e', 's': 'f', 't': 'g',
           'u': 'h', 'v': 'i', 'w': 'j', 'x': 'k',
           'y': 'l', 'z': 'm'}
    return dic[word]

while True:
    a_password = input('请输入文本(输明文，得密文；输密文，得明文):\n')
    r_password1 = 'qwertyuiopasdfghjklzxcvbnm'
    r_password2 = 'asdfqwerzxcvtyuiopghjklbnm'
    r_password3 = 'poiuytrewqasdfghjklmnbvcxz'
    if is_str(a_password, r_password1):
        print('对应的结果为:\n', simple_replace(a_password, r_password1, r_password2, r_password3))
        break
    else:
        pass
```