一、 XOR

1.作业基本信息

这个作业的目标	**1.实现Github上提交文件 2.接口的导入及其使用 3.文件格式的规范化**
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：利用异或运算实现加解密作业

2.2 设计思路

对于题目要求设计异或运算实现加解密的算法。

异或真值表如下：

R	S	P
0	0	0
0	1	1
1	0	1
1	1	0

选用异或运算的原因如下：

一个数字message在通过两次与同一数字key进行异或运算后，依然能得到原来的数字message

则运用此规则进行加密与解密操作。

加密操作：

首先将文件转换成二进制数，再生成与该二进制数等长的随机密钥，将二进制数与密钥进行异或操作，得到加密后的二进制数。

解密操作：

将加密后的二进制程序与密钥进行异或操作，就得到原二进制数，最后将原二进制数恢复成文本文件。

设计思路如下：

1. 生成随机密钥

对于密钥，可以是用户约定的一组口令，这种方法操作比较简单，直接进行输入即可，在本程序中，采用系统生成的随机密钥，可以较为便利解决选择密钥问题。

运用python的secrets库中的伪随机数模块，生成随机密钥，为了密钥与之后的数据能进行异或操作，需要对生成的随机密钥进行数据处理，将密钥处理为所需要的二进制数。

```
from secrets import token_bytes
from typing import Tuple

# 随机生成密钥
def random_key(length:int) -> int:
    key:bytes = token_bytes(nbytes=length)
    key_int:int = int.from_bytes(key, 'big')
    return key_int
```

2. 对传入数据进行加密

在encrypt函数中，传入str对象，返回元组，然后用encode，将字符串编码编码成字节串，再以同样的方式`int.from_bytes`将其转换为int二进制对象，然后用此二进制对象与所生成的随机密钥进行异或操作，得到加密文本。

```
# 编码密文
def encrypt(raw:str) -> Tuple[int, int]:
    raw_bytes:bytes = raw.encode()
    raw_int:int = int.from_bytes(raw_bytes, 'big')
    key_int:int = random_key(len(raw_bytes))
    return raw_int ^ key_int, key_int
```

3. 对于密文进行解密

对于decrypt的解密函数，需要对加密文本和密钥进行异或操作，计算出解密后的文本，之后再将解密后的二进制对象转换为字节串对象，再通过decode，将字节串转换为字符串。

```
# 解码密文
def decrypt(encrypted:int, key_int:int) -> str:
    decrypted:int = encrypted ^ key_int
    length = (decrypted.bit_length() + 7) // 8
    decrypted_bytes:bytes = int.to_bytes(decrypted, length, 'big')
    # decrypted_file=decrypted_bytes.decode()
    # print(decrypted_file)
    return decrypted_bytes.decode()
```

至此，对于异或操作进行加解密的程序算法已经列举在上，以下将描述关于文本存储和读取。

```
# 文件中读取并调用编码密文
def encrypt_file(path:str, key_path=None, *, encoding='utf-8'):
    path = Path(path)
    cwd = path.cwd() / path.name.split('.')[0]
    path_encrypted = cwd / path.name
    if key_path is None:
```

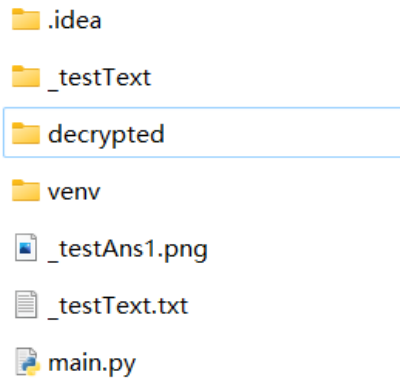
```
        key_path = cwd / 'key'
    if not cwd.exists():
        cwd.mkdir()
        path_encrypted.touch()
        key_path.touch()
    with path.open('rt', encoding=encoding) as f1, \
        path_encrypted.open('wt', encoding=encoding) as f2, \
            key_path.open('wt', encoding=encoding) as f3:
        encrypted, key = encrypt(f1.read())
        json.dump(encrypted, f2)
        json.dump(key, f3)

# 文件中得出并存储解码密文（明文）
def decrypt_file(path_encrypted:str, key_path=None, *, encoding='utf-8'):
    path_encrypted = Path(path_encrypted)
    cwd = path_encrypted.cwd()
    path_decrypted = cwd / 'decrypted'
    if not path_decrypted.exists():
        path_decrypted.mkdir()
        path_decrypted /= path_encrypted.name
        path_decrypted.touch()
    if key_path is None:
        key_path = cwd / 'key'
    key_path = cwd.cwd() / path_encrypted.name.split('.')[0] / 'key'
    path_decrypted = cwd / 'decrypted' / path_encrypted.name
    with path_encrypted.open('rt', encoding=encoding) as f1, \
            key_path.open('rt', encoding=encoding) as f2, \
            path_decrypted.open('wt', encoding=encoding) as f3:
        decrypted = decrypt(json.load(f1), json.load(f2))
        f3.write(decrypted)
```

3.部分单元测试展示

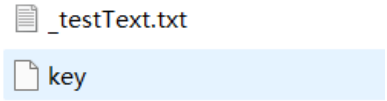
以下为对于密文的测试结果

```
数据为： abcdefg1234567888
密文为： 14866717205871052674593284738754677260178
密钥为： 25460172937930752775200693229427408963498
明文为： abcdefg1234567888
```

以下为对于处理文件的存储结果



以下为密文与密钥的存储文件



更多详细情况可以直接进行文件测试。

二、 Caesar

1.作业基本信息

这个作业的目标	1.实现Caesar密码加解密
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：利用Caesar密码加解密作业

2.2 设计思路

凯撒加密(Caesarcipher)是一种简单的消息编码方式：它根据字母表将消息中的每个字母移动常量位k。

选用凯撒加解密的原因如下：

凯撒加密后的message整体偏移k个单位，解密的时候返回k个单位即可。

则运用此规则进行加密与解密操作。

设计思路如下：

1. 对传入数据进行加密

首先对传入的数据字符串转为列表，然后转ASCII码，对其进行k个单位的偏移

```
# 加密
def caesar(message):
    message1 = message.upper()  # 把明文字母变成大写
    message1 = list(message1)  # 将明文字符串转换成列表
    list1 = []
    for i in range(len(message1)):
        if message1[i] == ' ':
            list1.append(message1[i])  # 若为空格不用移动
        elif ord(message1[i]) <= 90 - 3 + 1:  # A-X右移三位
            list1.append(chr(ord(message1[i]) + 3))
            result = ''.join(list1)  # 列表转换成字符串
        else:
            list1.append(chr(ord(message1[i]) - (26 - 3)))  # Y和Z回到A、B
            result = ''.join(list1)
    return result
```

2. 对于密文进行解密

解密则进行反向操作。

3.部分单元测试展示

以下为对于密文的测试结果

```
input code:
abcde1234578
encode: DEFGH45678:;
decode: ABCDE1234578
```

更多详细情况可以直接进行文件测试。

三、 Hill

1.作业基本信息

这个作业的目标	1.实现Hill密码加解密
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：利用Hill密码加解密作业

2.2 设计思路

希尔密码，是运用基本矩阵论原理的替换密码，由Lester S. Hill在1929年发明。

首先确定一个密钥，这个密钥为n × n的矩阵，并且必须是可逆的。然后将字符与数字创建映射表，一个字符对应一个数字。加密时，将明文分割为n个字符一组，在映射表中找到字符对应数字，每一组字符就变为一个1 × n 的矩阵，将这个矩阵与密钥相乘的结果模72（字典数据数目），再在映射表中找数字对应字符转化，就得到了密文。解密时，先将字符转数字，n个一组，形成1 × n 的矩阵，把这些矩阵与密钥的逆矩阵分别相乘，左乘还是右乘需要与之前一致，将乘得的结果加72的倍数转为正数后模72，将这些数字通过映射表转为明文。

设计思路如下：

在本程序中，仅用字母作为字典

1. 对传入数据进行加密

- 输入数据后，将字母与字典替代，对应数字并准备化为矩阵；
- 在输入约定的密钥矩阵时，判断是否可逆；
- 密钥矩阵与输入数据的矩阵进行矩阵乘法，得出密文。

2. 对于密文进行解密

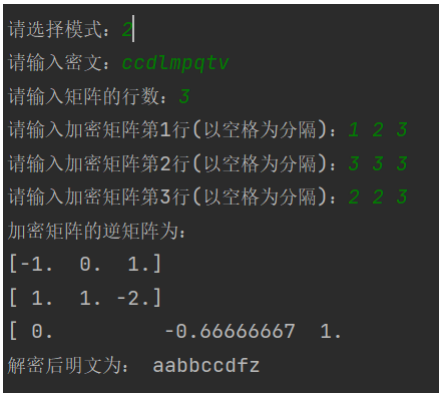
解密则运用约定的密钥矩阵，先求出其逆矩阵，然后与密文进行线代的矩阵乘法，求解出明文。

需要注意的是，由于密钥矩阵与数据矩阵并不一定能够正好进行，矩阵的乘法运算（表现为密钥矩阵是 n X n 的，但数据矩阵可能并不能正好整除）此时数据方面需要添加一定冗余字符。

3.部分单元测试展示

以下为对于密文的测试结果

```
========Hill密码========

1.加密
2.解密

注意：如果输入矩阵的逆矩阵中含有小数，采用四舍五入的方法

请选择模式：1
请输入明文：aabbccdf
请输入矩阵的行数：3
请输入加密矩阵第1行(以空格为分隔)：1 2 3
请输入加密矩阵第2行(以空格为分隔)：3 3 3
请输入加密矩阵第3行(以空格为分隔)：2 2 3

添加了1个z补全明文
加密后密文为： ccdlmpqtv
```



更多详细情况可以直接进行文件测试。

四、 Des

1.作业基本信息

这个作业的目标	**1.实现Github上提交文件** **2.接口的导入及其使用** **3.文件格式的规范化**
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：1. 网上搜索DES的源代码。
>
> 2. 利用DES源代码实现下面功能：
> 3. 给定某个Sbox的输入差分情况下，计算所有输入对和所有Sbox输出差分的分布情况
> 4. 统计DES算法在密钥固定情况，输入明文改变1位、2位，。。。64位时。输出密文位数改变情况。
> 5. 统计DES算法在明文固定情况，输入密钥改变1位、2位，。。。64位时。输出密文位数改变情况。
>
> 为了具有客观性，2，3小题需要对多次进行统计，并计算其平均值。

2.2 设计思路

2.2.1 原理与理论基础

DES算法为对称算法中的分组加密算法

1. IP置换 按照一定规则，将原来的64位二进制位重新排序
2. 轮函数 E扩展置换 将32位输出扩展为48位输出

3. 轮函数 S盒压缩处理 经过扩展的48位明文和48位密钥进行异或运算后，再使用8个S盒压缩处理得到32位数据。再将48位输入等分为8块，每块6位输出压缩位4位输出。
4. 轮函数 P盒置换 S盒所得结果再经过P盒置换，至此，一次轮函数操作完毕。

2.2.2 内容与实现过程

des通过明文和密钥加密
明文 密钥都为64位

加密流程：

密钥部分

1、将64位密钥通过pc1置换成56位密钥
2、用56位密钥移位生成16个56位子密钥
3、将16个56位子密钥通过pc2置换成16个48位子密钥，等待加密时使用（K1,K2,K3...）

加密部分

1、将64位明文进行ip置换
2、截取前32位为L0, 后32位为R0
3、进行轮运算

第一轮
L1 = R0
R1 = L0 ^ f(R0, K1)
第二轮
L2 = R1
R2 = L1 ^ f(R1, K2)
...
f为 f(R, K)
{
let E_R = e(R);
let temp = xor(E_R, K);
return s(temp);
}
首先将32位的R扩展成48位，
然后将48位的R和48位K进行异或，
最后进行S盒替换，变成32位的文本。
f为 S(text)
首先将48位的text分割成8个6位的数，
然后每个6位的数对应一个S盒，
将6位数的第一位和第六位组成行号，中间四位组成列号，置换成相应S盒的数，
然后按S0S1S2...变成32的文本输出
4、将16轮变化后的L16,R16变成R16L16
5、对R16L16进行ip逆置换

3.部分单元测试展示

可以看到其位数变化较为平稳

```
输入改变类型(0:明文固定，密匙改变;1:密匙固定，明文改变)
0
统计次数
5
密匙 : 00111111001000110101001101001011010110110000000001101101010011111
明文 : 00100111011010010011101001100111000101000000010100111100010100000
密匙改变1位 : 10111111001000110101001101001011010110110000000001101101010011111    密文改变32位
密匙改变2位 : 11111111001000110101001101001011010110110000000001101101010011111    密文改变27位
密匙改变3位 : 11011111001000110101001101001011010110110000000001101101010011111    密文改变32位
密匙改变4位 : 11001111001000110101001101001011010110110000000001101101010011111    密文改变27位
```

```
密匙改变56位 : 10010011110110111011111111010000111110011111100001100001100011100    密文改变33位
密匙改变57位 : 10010011110110111011111111010000111110011111100001100001110011100    密文改变38位
密匙改变58位 : 10010011110110111011111111010000111110011111100001100001111011100    密文改变32位
密匙改变59位 : 10010011110110111011111111010000111110011111100001100001111111100    密文改变31位
密匙改变60位 : 10010011110110111011111111010000111110011111100001100001111101100    密文改变35位
密匙改变61位 : 10010011110110111011111111010000111110011111100001100001111100100    密文改变36位
密匙改变62位 : 10010011110110111011111111010000111110011111100001100001111100000    密文改变28位
密匙改变63位 : 10010011110110111011111111010000111110011111100001100001111100010    密文改变31位
密匙改变64位 : 10010011110110111011111111010000111110011111100001100001111100011    密文改变31位
第1次统计31.625位
第2次统计32.188位
第3次统计31.922位
第4次统计32.562位
第5次统计31.828位
```

```
输入改变类型(0:明文固定，密匙改变;1:密匙固定，明文改变)
1
统计次数
5
密匙 : 00101100011111010110001000100001011100010111000000111010011101l0
明文 : 00110110011000010000011100011100010010110011000000001111100111l0
明文改变1位 : 10110110011000010000011100011100010010110011000000001111100111l0    密文改变34位
明文改变2位 : 11110110011000010000011100011100010010110011000000001111100111l0    密文改变33位
明文改变3位 : 11010110011000010000011100011100010010110011000000001111100111l0    密文改变37位
明文改变4位 : 11000110011000010000011100011100010010110011000000001111100111l0    密文改变30位
明文改变5位 : 11001110011000010000011100011100010010110011000000001111100111l0    密文改变33位
```

```
明文改变57位 : 10110110100011101011111011111100111010001011000010111010111111l1    密文改变24位
明文改变58位 : 10110110100011101011111011111100111010001011000010111010101111l1    密文改变32位
明文改变59位 : 10110110100011101011111011111100111010001011000010111010100111l1    密文改变37位
明文改变60位 : 10110110100011101011111011111100111010001011000010111010100011l1    密文改变29位
明文改变61位 : 10110110100011101011111011111100111010001011000010111010100001l1    密文改变30位
明文改变62位 : 10110110100011101011111011111100111010001011000010111010100000l1    密文改变33位
明文改变63位 : 10110110100011101011111011111100111010001011000010111010100001l1    密文改变35位
明文改变64位 : 10110110100011101011111011111100111010001011000010111010100000l0    密文改变32位
第1次统计32.281位
第2次统计32.453位
第3次统计32.078位
第4次统计32.672位
第5次统计31.719位
```

结果符合题意.

4. 补充

> 分析并解释 《中华人民共和国电子签名法(2019修正)》修改的内容

第十一条

当事人提交的电子数据，通过电子签名、可信时间戳、哈希值校验、区块链等证据收集、固定和防篡改的技术手段或者通过电子取证存证平台认证，能够证明其真实性的，互联网法院应当确认。

影响：互联网法院认可真实的电子数据作为案件审理的证据，这些电子数据可以通过电子签名、可信时间戳、哈希值校验、区块链等技术手段进行真实性确认。

更多详细情况可以直接进行文件测试。

五、 DH

1.作业基本信息

这个作业的目标	**1.实现Github上提交文件** **2.接口的导入及其使用** **3.文件格式的规范化**
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：编程实现 DH 密钥协商协议
> 共同参数。素数P，P的一个生成元g
>
> 1. A 随机选择一个 [1,p-1] 范围内的数`x`，计算 `X` = g^x
> 2. B 随机选择一个 [1,p-1] 范围内的数`y`，计算 `Y` = g^y mod p, 并将结果发送给A。
>   k` = Y^x = X^y

2.2 设计思路

2.2.1 原理与理论基础

Diffie-Hellman密钥交换算法

- 有两个全局公开的参数，一个素数p和一个整数a，a是p的一个原根（对于正整数gcd(a,m)=1，如果a是模m的原根，那么a是整数模m乘法群的一个生产元）；
- 假设用户A和B希望交换一个密钥，用户A选择一个作为私有密钥的随机数XA<p，并计算公开密钥YA = a^XA mod p，A对XA的值保密存放而使YA能被B公开获得。类似地，用户B选择一个私有的随机数XB<p，并计算公开密钥YB = a^XB mod p。B对XB的值保密存放而使YB能被A公开获得。
- 用户A产生共享秘密密钥的计算方式是K = (YB)^XA mod p。同样，用户B产生共享秘密密钥的计算是K = (YA)^XB mod p。这两个计算产生相同的结果

2.2.2 内容与实现过程

1. 输入素数 p

此时将对输入的 p 进行素数的判断

使用 `isPrime(int n)`

2. 求素数 p 的一个生成元 g

当a是p的原根，就是a^(p - 1) = 1(mod p)当且仅当指数为p-1的时候成立，则取a在[2, p-1]之间进行取值，若是其中有满足上述式子的a则放到列表中，最后在自动生成原根的时候是人为的取得最大的那个数值作为原根。

例如当p = 7时，a从2开始取值，因为2^3 = 1(mod 7),3!=6,所以2不是p的原根；

当a = 3时,3^1 = 3(mod 7),3^2 = 3(mod 7),3^3 = 6(mod 7),3^4 = 4(mod 7),3^5 = 5(mod 7),3^6 = 1(mod 7)

所以p = 7的一个原根就是3，同样的道理可以求出p的所有的原根。

使用函数 `get_generator(int p)`

```
def get_generator(p):
    # 得到所有的原根
    # 寻找生成元a
    a = 2
    list = []
    while a < p:
        flag = 1
        while flag != p:
            if (a ** flag) % p == 1:
                break
            flag += 1
        if flag == (p - 1):
            list.append(a)
        a += 1
    return list
```

3. A与B得到通过随机生成的私钥，范围在[1,p-1]
4. 通过素数p，生成元，密钥 生成交换的密钥。
5. 进行计算交换得到的密钥与自己的密钥，得到共享秘密密钥k

以下为部分功能代码：

- GCD 求两数最大公因子

```
def GCD(a, b):
    if (b):
        return GCD(b, a % b)
    else:
        return a
```

- 费马检测

```
def ExpMod(b, n, m):
    return b ** n % m
```

- 判断 g 为模 p 乘的生成元 `bool isPrimeRoot(g, p)`

在上述功能描述中包含选用生成元的过程。

寻找p的生成元，从 g 从2，3等较小的数开始进行穷举。 计算 g^n mod p， 1<=n < p-1 的到的数据不同，g即为生成元。

3.部分单元测试展示

以下为求两个数最大公因子的展示

```
Please input gcd num1: 30
Please input gcd num2: 29
gcd: 1
```

以下为DH算法的完整结果展示：

```
Please input your number(It must be a prime!): 79
79 is a prime!
79 的一个原根为：77
-----------------------------------------------------
A随机生成的私钥为：7
B随机生成的私钥为：43
-----------------------------------------------------
A的计算数,用于交换的密钥为：30
B的计算数,用于交换的密钥为：63
-----------------------------------------------------
A通过交换密钥计算后的共享秘密密钥为：66
B通过交换密钥计算后的共享秘密密钥为：66
--------------------------True or False---------------
判断在交换密钥后,经过计算过后的,在A,B各自端的共享秘密密钥是否相等
True
```

通过输入素数，找到其一个生成元，A与B各自随机生成密钥，通过素数计算出用于交换的密钥，交换密钥后计算出共享秘密密钥。

更多详细情况可以直接进行文件测试。

六、 RSA

1.作业基本信息

这个作业的目标	**1.实现Github上提交文件** **2.接口的导入及其使用** **3.文件格式的规范化**
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：编程实现RSA基本算法
> 计算： 在一个RSA系统中，一个给定用户的公钥 e = 31, n = 3599.求这个用户的私有密钥。

2.2 设计思路

2.2.1 原理与理论基础

RSA公开密钥密码体制的原理是：根据数论，寻求两个大素数比较简单，而将它们的乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥.

RSA算法的具体描述如下：

（1）任意选取两个不同的大素数p和q计算乘积 `n=pq` ，`phi(n)=(p-1)(q-1)`

（2）任意选取一个大整数e，满足 `gcd(e,phi(n))=1` ，整数e用做加密钥（注意：e的选取是很容易的，例如，所有大于p和q的素数都可用）

（3）确定的解密钥d，满足 `(d * e) mod phi(n)=1` ，即 `(d * e)=k*phi(n)+1, k>=1` 是一个任意的整数；所以，若知道 e和 phi(n)，则很容易计算出 d

（4）公开整数 n 和 e ， 秘密保存 d

（5）将明文 m（m<n是一个整数）加密成密文 c ， 加密算法为 `c=E(m)= m^e mod n`

（6）将密文c解密为明文m，解密算法为 `m=D(c)= c^d mod n`

然而只根据n和e（注意：不是p和q）要计算出d是不可能的。因此，任何人都可对明文进行加密，但只有授权用户（知道d）才可对密文解密 。

2.2.2 内容与实现过程

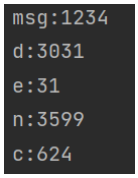
1. 确定 p, q 并求出 n
2. 给出 e 并判断是否为素数
3. 通过算法求出逆元 d ,即作为私钥

```
    def ex_gcd(self, a, b, d, x, y):
        '''
        函数结束时，（x + b) % b为 (a % b)的乘法逆元
        '''
        if b == 0:
            d[0], x[0], y[0] = a, 1, 0
        else:
            self.ex_gcd(b, a % b, d, y, x)
            y[0] -= a // b * x[0]
```

4. 进行加密与解密操作,通过 m 得到 c ,或是 c 得到 m

3.部分单元测试展示

以下为使用题目数据得出的结果.



结果符合题意,结果正常.

更多详细情况可以直接进行文件测试。

七、 Hash

1.作业基本信息

这个作业的目标	**1.实现Github上提交文件** **2.接口的导入及其使用** **3.文件格式的规范化**
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：利用软件提供的Hash函数实现查找本地硬盘重复文件。编程并撰写报告。
> 提示：
>
> 1. 首先计算硬盘文件的Hash，并记录文件路径与Hash值的对应关系
> 2. 利用具有相同Hash值的文件，内容就极其可能相同的特性，查找具有相同Hash值的文件。
> 3. 可以简单比较一下具有相同Hash值的文件是否长度，部分随机内容相同，如果相同则直接输出，否则说明发生了碰撞。
> 4. 注意保护个人隐私。

2.2 设计思路

2.2.1 原理与理论基础

散列函数（或散列算法，又称哈希函数，英语：Hash Function）是一种从任何一种数据中创建小的数字"指纹"的方法。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来。该函数将数据打乱混合，重新创建一个叫做散列值（hash values，hash codes，hash sums，或hashes）的指纹。散列值通常用一个短的随机字母和数字组成的字符串来代表。好的散列函数在输入域中很少出现散列冲突。在散列表和数据处理中，不抑制冲突来区别数据，会使得数据库记录更难找到。

加密散列函数，是散列函数的一种。它被认为是一种单向函数，也就是说极其难以由散列函数输出的結果，回推输入的资料是什么。这种散列函数的输入资料，通常被称为讯息（message），而它的输出结果，经常被称为讯息摘要（message digest）或摘要（digest）

MD5算法的原理可简要的叙述为：MD5码以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。

2.2.2 内容与实现过程

通过python的hashlib实现

3.部分单元测试展示

以下为使用MD5算法计算散列值,比较两个文件的散列值,对照分析得出文件是否为重复.

以下为不同的文件,即进行了修改过后的文件:

```
file1:123456789
file2:123556789
file1_md5: 25f9e794323b453885f5181f1b624d0b
file2_md5: 1d675c2be727069e73153d1b1b9c3e34
file has changed
```

以下为内容相同的文件:

```
file1:123456789
file2:123456789
file1_md5: 25f9e794323b453885f5181f1b624d0b
file2_md5: 25f9e794323b453885f5181f1b624d0b
file not changed
```

更多详细情况可以直接进行文件测试。

八、 RSA与HASH 认证

1.作业基本信息

这个作业的目标	**1.实现Github上提交文件** **2.接口的导入及其使用** **3.文件格式的规范化**
Github仓库	*仓库链接*
其他参考文献	...

2.计算模块接口的设计与实现过程

2.1 问题需求

> 题目：编程，利用RSA， Hash 实验数字签名和验证。
>
> 1. 选择语言提供的Hash函数，计算数据的Hash值
> 2. 用私钥对 Hash值 进行签名，
> 3. 用公钥对签名值进行验证。
> 4. 网上搜集资料，分析并解释 《中华人民共和国电子签名法(2019修正)》修改的内容。md格式，用git管理。

2.2 设计思路

2.2.1 原理与理论基础

RSA公开密钥密码体制的原理是：根据数论，寻求两个大素数比较简单，而将它们的乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥.

RSA算法的具体描述在此省略.

散列函数（或散列算法，又称哈希函数，英语：Hash Function）是一种从任何一种数据中创建小的数字"指纹"的方法。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来。该函数将数据打乱混合，重新创建一个叫做散列值（hash values，hash codes，hash sums，或hashes）的指纹.

在本题中采用的散列算法为MD5.

1. RSA加密过程简述

A和B进行加密通信时,B首先要生成一对密钥。一个是公钥，给A，B自己持有私钥。A使用B的公钥加密要加密发送的内容，然后B在通过自己的私钥解密内容

2. 假设A要想B发送消息，A会先计算出消息的消息摘要，然后使用自己的私钥加密这段摘要加密，最后将加密后的消息摘要和消息一起发送给B，被加密的消息摘要就是“签名”。

B收到消息后，也会使用和A相同的方法提取消息摘要，然后使用A的公钥解密A发送的来签名，并与自己计算出来的消息摘要进行比较。如果相同则说明消息是A发送给B的，同时，A也无法否认自己发送消息给B的事实。

其中，A用自己的私钥给消息摘要加密成为“签名”；B使用A的公钥解密签名文件的过程，就叫做“验签”。

3. 签名过程

1）A提取消息m的消息摘要h(m),并使用自己的私钥对摘要h(m)进行加密,生成签名s

2）A将签名s和消息m一起,使用B的公钥进行加密,生成密文c,发送给B。

具体：

- A计算消息m的消息摘要,记为 h(m)
- A使用私钥(n,d)对h(m)加密，生成签名s ,s满足：`s=(h(m))^d mod`;

由于A是用自己的私钥对消息摘要加密，所以只用使用s的公钥才能解密该消息摘要，这样A就不可否认自己发送了该消息给B。

A发送消息和签名(m,s)给B。

4. 验证过程

1）B接收到密文c,使用自己的私钥解密c得到明文m和数字签名s

2）B使用A的公钥解密数字签名s解密得到H(m).

3）B使用相同的方法提取消息m的消息摘要h(m)

4）B比较两个消息摘要。相同则验证成功;不同则验证失败。

具体：

1.B计算消息m的消息摘要,记为h(m);

2.B使用A的公钥(n,e)解密s,得到 `H(m) = s^e mod n`;

3.B比较H(m)与h(m),相同则证明

2.2.2 内容与实现过程

1. 通过MD5算法计算数据散列值，在此采用以下数据散列值进行计算。

```
file1:123456789
file2:123456789
file1_md5: 25f9e794323b453885f5181f1b624d0b
file2_md5: 25f9e794323b453885f5181f1b624d0b
file not changed
```

2. 由于数据散列值对于本题中的算法数据要求可能过大，在此选用前五位散列值进行计算，即`25f9e`
3. 本题中选用加密算法为RSA算法，采用 `p=6007`，`q=360089`，`e=795479`
4. 求出私钥 `d=1026474959`
5. 在此将散列值化为ASCII，方便后续计算，并对ASCII进行了一定处理.

```
    msg = map(ord,"25f9e")
    msg = list(map(lambda x:str(x-30),list(msg)))
    msg = ''.join(msg)
    msg = int(msg)
```

6. 转回消息的处理.

```
    m = re.findall(r'.{2}', str(m))
    m = map(int,m)
    m = map(lambda x:x+30,m)
    m = "".join(map(chr,m))
```

对应上述过程，散列值则对应消息摘要，A与B各自拥有一对密钥。

7. 得到A的消息摘要h(m)后，B对于得到的消息明文m的进行生成散列值，同样得到相同的散列值，即B得到的消息摘要H(m)=`25f9e`，二者相同，则说明了此信息为A所发.

3.部分单元测试展示

以下为使用试用数据得出的结果.

```
e:795479
d:1026474959
msg:25f9e
c:473193656
```

结果符合题意.

4. 补充

> 分析并解释 《中华人民共和国电子签名法(2019修正)》修改的内容

第十一条

当事人提交的电子数据，通过电子签名、可信时间戳、哈希值校验、区块链等证据收集、固定和防篡改的技术手段或者通过电子取证存证平台认证，能够证明其真实性的，互联网法院应当确认。

影响：互联网法院认可真实的电子数据作为案件审理的证据，这些电子数据可以通过电子签名、可信时间戳、哈希值校验、区块链等技术手段进行真实性确认。

更多详细情况可以直接进行文件测试。