上机常用代码

(1) 输入代码

(2) 矩阵乘法 (np.dot(A, B))

(2.2) 矩阵转置

(3) 生成 nxn 的 空矩阵

(4) 字典排序

(5) 统计列表中所有元素出现的频率

(6) list和set相互转换

(7) 列表转整数

(8) 数字字符串 转整数

(9) 读取二进制文件

(9.1) 读二进制文件

(10) 判断素数 和 埃氏筛选法

(11) 最大公约数

(12) format格式化输出

(13) 全排序

(13) 插入排序

(14) 删除list里面的重复元素 (位置不变)

(15) 遍历列表时删除元素的正确做法

(16) 集合操作

(17) 分割连续字符串

(18) 用 字符串 分割字符串

(19) maketrans, translate 【字符串对应地转换】

(20) 日期计算

(21) 方差

苏大python习题

1. 给定整数m和n，如果m和n都大于1，则判定m和n是否互质，并返回判定结果。

2. 一个整数列表L=[a1, a2, ..., an]中，如果一对数(ai, aj)满足ai>aj 且 i<j，那么这对数就称为一个逆序，列表L中逆序的数量称为逆序数。求一个整数列表L的 逆序数。

3. 矩阵相乘：

4. 一维列表转成二维列表： 输入一个长度为 n*n 的一维列表， 返回一个n行n列的二维列表。

5. 统计字符串

6. Jaccard系数

7. 统计最多的字符次数

8. 数字和平均值

苏大真题[2005-2019]

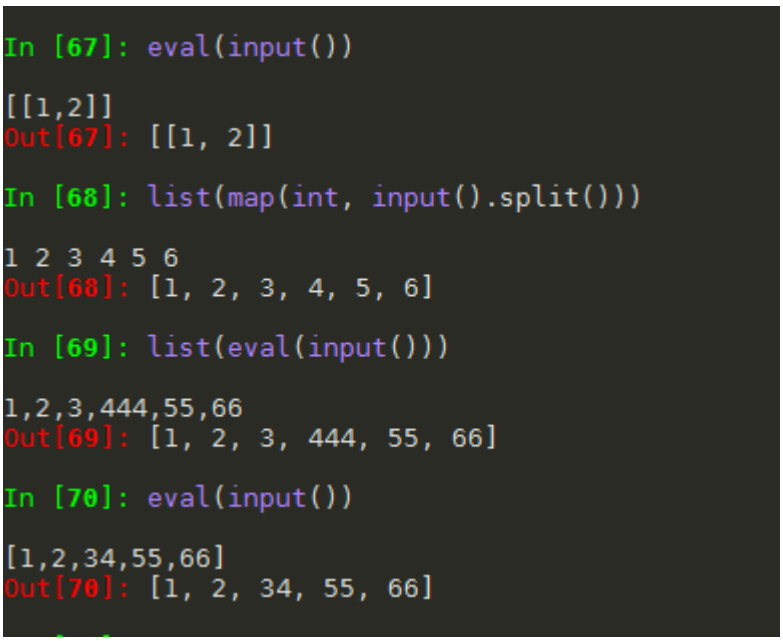
2005. 把一个数表示成若干个素数的和.

2005.2 统计篇文章中各英文字母的个数，并排序.

2006. 找出 100 到 1000 内的不含 9 的素数，存到 result 文件中.

上机常用代码

(1) 输入代码



(2) 矩阵乘法 (np.dot(A, B))

```
def matrixMul(A, B):
    if len(A[0]) == len(B):  # A列数=B行数
        res = [[0] * len(B[0]) for i in range(len(A))]  # 生成 A行 x B列的0矩阵
        for i in range(len(A)):
            for j in range(len(B[0])):
                for k in range(len(B)):
                    res[i][j] += A[i][k] * B[k][j]
        return res
    return ("输入矩阵有误!")

matrixMul(mat1, mat2)
Out[13]: [[19, 22], [43, 50]]

mat1
Out[14]: [[1, 2], [3, 4]]

mat2
Out[15]: [[5, 6], [7, 8]]
```

```
In [12]: def matrixMul(A, B):
    ...:     if len(A[0]) == len(B):  # A列数=B行数
    ...:         res = [[0] * len(B[0]) for i in range(len(A))]
    ...:         for i in range(len(A)):
    ...:             for j in range(len(B[0])):
    ...:                 for k in range(len(B)):
    ...:                     res[i][j] += A[i][k] * B[k][j]
    ...:         return res
    ...:     return ("输入矩阵有误!")
    ...:

In [13]: matrixMul(mat1, mat2)
Out[13]: [[19, 22], [43, 50]]

In [14]: mat1
Out[14]: [[1, 2], [3, 4]]

In [15]: mat2
Out[15]: [[5, 6], [7, 8]]
```

(2.2) 矩阵转置

```
matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

list(zip(*matrix))
Out[34]: [(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]

matrix
Out[42]: [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]

[[row[i] for row in matrix] for i in range(4)]
Out[43]: [[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

(3) 生成 nxn 的 空矩阵

```
# 生成 nxn 的 空矩阵
res = [[] for i in range(0, n)]
```

(4) 字典排序

```
# 按value排序(降序)，如果value相同则按key排序(字典序升序)
freq = sorted(freq.items(), key=lambda x:(-x[1], x[0]))
```

(5) 统计列表中所有元素出现的频率

```
from collections import Counter

a = "i love love you you"

dict( Counter(list(a.split())) )

Out[103]: {'i': 1, 'love': 2, 'you': 2}
```

或者：

```
from collections import defaultdict
frequences = defaultdict(int)
print(frequences)
Out[37]: defaultdict(int, {})

for e in a.split():
    frequences[e] += 1

frequences
Out[39]: defaultdict(int, {'i': 1, 'love': 2, 'you': 2})

s + t - s
Out[112]: Counter({'c': 1, 'b': 2, 'a': 1, 'd': 1})
list((s + t - s).elements())
Out[115]: ['c', 'b', 'b', 'a', 'd']
```

(6) list和set相互转换

```
set([1,2,3,4,4,5,5])
{1,2,3,4,5}
list(set([1,2,3,4,5,5]))
[1,2,3,4,5]
```

(7) 列表转整数

```
a = [1,2,3]

int("".join(list(map(str, a))))
Out[109]: 123
```

(8) 数字字符串 转整数

```
int('0e00', 16)     # 按照16进制转换成10进制
int('001', 2)       # 按照二进制方式转换成10进制
int('012', 8)       # 按照八进制方式转换成10进制
```

↓	2进制	8进制	10进制	16进制
2进制	-	bin(int(x, 8))	bin(int(x, 10))	bin(int(x, 16))
8进制	oct(int(x, 2))	-	oct(int(x, 10))	oct(int(x, 16))
10进制	int(x, 2)	int(x, 8)	-	int(x, 16)
16进制	hex(int(x, 2))	hex(int(x, 8))	hex(int(x, 10))	-

bin()、oct()、hex()的返回值均为字符串，且分别带有0b、0o、0x前缀。

(9) 读取二进制文件

```
def main():
    res = [random.randrange(0, 9999, 1) for i in range(0, 32)]
#    print(res)

#    with open('./file/2010.txt', 'rb+') as f:
#        for i in res:
#            s = struct.pack('i', i)
#            f.write(s)

    # 写多少数据，一定要弄清楚
    len1 = len(res)
    nums = []
    with open('./file/2010.txt', 'rb+') as f:
        for i in range(len1):
            data = f.read(4)
            elem = struct.unpack('i', data)[0]
            nums.append(elem)

    # 按照 16进制 转换成 10进制
```

```
    nums.sort(reverse=True)

    print("数据:", nums)
```

(9.1) 读二进制文件

```
def read_data():
    data = []
    with open('./file/file_2017_2.txt', 'rb') as f:
        num = f.read(4)
        while num:
            data.append(int.from_bytes(num, byteorder='little'))
            num = f.read(4)
    return data
```

(10) 判断素数 和 埃氏筛选法

```
def is_prime(num):
    if num < 2:
        return False
    top = int(math.sqrt(num))
    i = 2
    while i <= top:
        if num % i == 0:
            return False
        i = i + 1
    return True

# 埃氏筛选法
def judge():
    n = int(input())
    res = 0

    is_prime = [True]*(n+1)
    is_prime[0] = is_prime[1] = True

    # 筛选出所有素数
    primes = []
    for i in range(2, n+1):
        if is_prime[i]:
            primes.append(i)
```

```
            j = 2*i
            while j <= n:               # 将 i 的倍数全部划去
                is_prime[j] = False
                j += i

    for i in range(2, n - 1):
        if is_prime[i] and is_prime[i+2]:
            res += 1
    print(res)

judge()
```

(11) 最大公约数

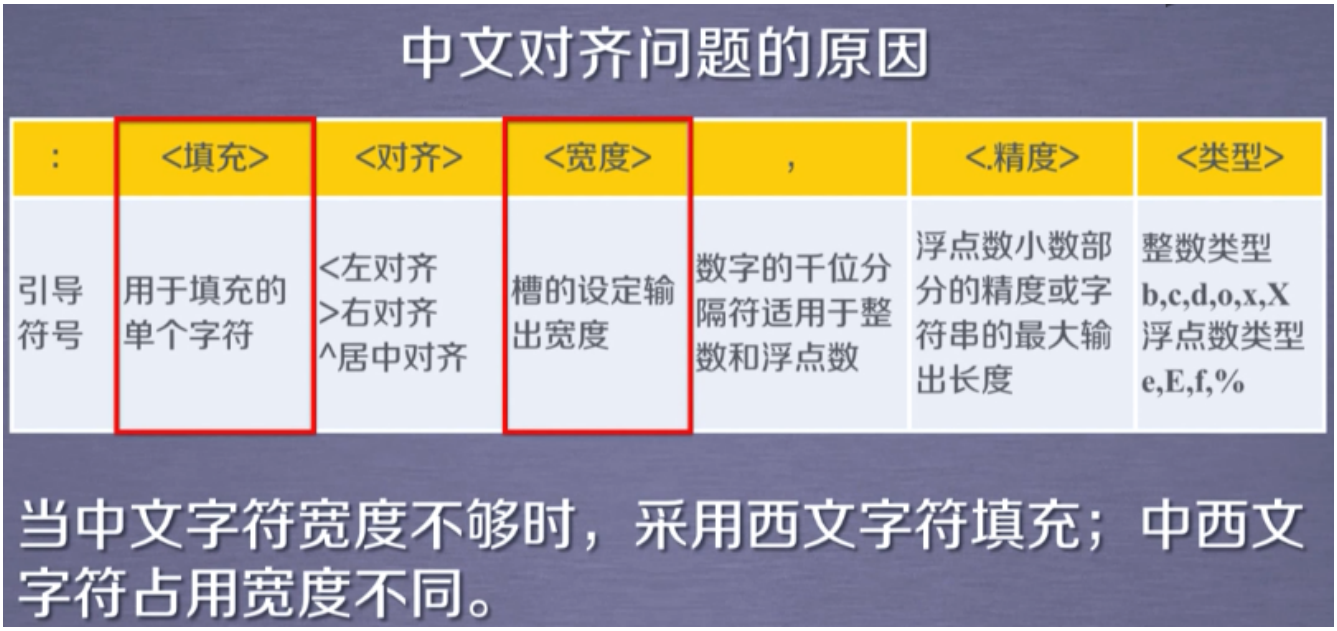
```
def gcd(num1, num2):
    if num1 < num2:
        num1, num2 = num2, num1

    while (num1 % num2):
        temp = num1 % num2
        num1 = num2
        num2 = temp

    return num2
```

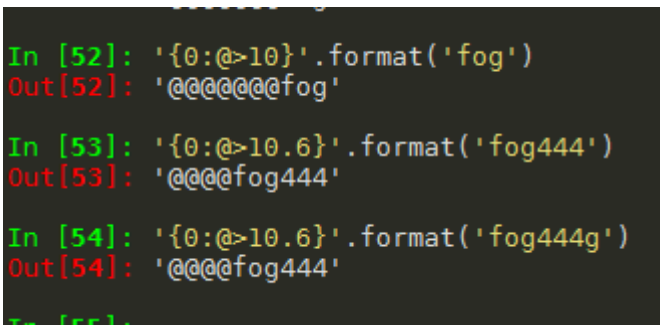
(12) format格式化输出

```
    for i in range(0, wlen):
        print('{0:8}'.format(ascii_words[i]), end='')
        if (i+1) % 10 == 0:
            print('')
```



题评析

format()方法的格式控制的语法格式如下：
{<参数序号>:<格式控制标记>}
格式控制标记包括：<填充><对齐><宽度><,><.精度><类型>等六个字段，这些字段都是可选的，可以组合使用。
填充常跟对齐一起使用，^、<、>分别是居中、左对齐、右对齐，后面带宽度，引导符号':'后面带填充的字符，只能是一个字符，不指定的话默认是用空格填充。
所以填充字符为@，宽度为10，精度为6，也就是字符串最大输出长度为6。Fog长度为3，可以正常输出。>表示右对齐，所以输出结果为@@@@@@@Fog。



(13) 全排序

```
def permute(L):
    if not L:
        return [L]
    else:
        res = []
        for i in range(len(L)):
            # 任一元素 + 剩余n-1个元素的全排序
            rest = L[:i] + L[i+1:]
            for x in permute(rest):
                res.append(L[i:i+1] + x)
        return res


permute([1,2,3])
```

```
Out[9]: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2,
1]]
```

或者：

```
def perm(L, cur):
    if cur == 3:
        print(L)
        return
    for i in range(1, 4):
        if not vis[i]:
            L[cur] = i
            vis[i] = True
            perm(L, cur + 1)
            vis[i] = False


L = [0 for i in range(0, 3)]
vis = [0 for i in range(0,10)]

perm(L, 0)
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```

(13) 插入排序

```
# -*- coding: utf-8 -*-
import random

# 找插入位置，使L仍保持升序
def find_insert_index(L, t):
    for i, x in enumerate(L):
        if x > t:
            return i
    return len(L)

A = [random.randint(0, 100) for i in range(10)]
```

```
print(A)
L = []
for x in A:
    i = find_insert_index(L, x)
    L.insert(i, x)
print(L)
```

(14) 删除list里面的重复元素 (位置不变)

```
l1 = ['b','c','d','c','a','a']
l2 = sorted(set(l1),key=l1.index)
print(l2)

l1 = ['b','c','d','c','a','a']
l2 = []
for i in l1:
    if not i in l2:
        l2.append(i)
print(l2)
```

(15) 遍历列表时删除元素的正确做法

```
21.Python-遍历列表时删除元素的正确做法
遍历在新在列表操作，删除时在原来的列表操作
a = [1,2,3,4,5,6,7,8]
print(id(a))
print(id(a[:]))
for i in a[:]:
    if i<5:
        a.remove(i)
    print(a)
print('-----------')
print(id(a))
```

(16) 集合操作

```
# 集合操作
a_set = set([1,2,3,4,7,8])
b_set = set([8,9,10,11,12,13])
```

```
print(a_set | b_set)        # 并集
print(a_set.union(b_set))   # 并集
print(a_set & b_set)        # 交集
print(a_set.intersection(b_set))  # 交集
print(a_set.difference(b_set))    # 差集
print(a_set - b_set)

print(a_set.symmetric_difference(b_set))   # 对称差
print(a_set ^ b_set)       # 对称差
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13}
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13}
{8}
{8}
{1, 2, 3, 4, 7}
{1, 2, 3, 4, 7}
{1, 2, 3, 4, 7, 9, 10, 11, 12, 13} n0
{1, 2, 3, 4, 7, 9, 10, 11, 12, 13}
```

(17) 分割连续字符串

```
data = 'TTTTThhiiiis isssss a   tesssst CAaaa as'

[i+j for i,j in re.findall(r'([a-zA-Z\s])(\1*)', data)]
Out[47]:
['TTTTT', 'hh', 'iiii', 's', ' ', 'i', 'sssss', ' ', 'a', '   ', 't',
'e', 'ssss', 't',' ','C','A','aaa',' ','a','s']

data = '5T2h4is i5s a3 te4st CA3a as10Z'

re.findall('(\d*)([a-zA-Z\s])', data)
Out[50]:
[('5', 'T'),('2', 'h'),('4', 'i'),('', 's'),('', ' '),('', 'i'),('5',
's'),('', ' '),
 ('', 'a'),('3', ' '),('', 't'),('', 'e'),('4', 's'),('', 't'),('', ' '),
('', 'C'),
 ('', 'A'),('3', 'a'),('', ' '),('', 'a'),('', 's'),('10', 'Z')]
```

(18) 用 字符串 分割字符串

```
s = 'iloveappleclass'

s.partition('apple')
Out[80]: ('ilove', 'apple', 'class')
```

(19) maketrans, translate 【字符串对应地转换】

```
table = ''.maketrans('abcdef123', 'uvwxyz@#$')
s = 'Python is a greate programming language. I like it!'
s.translate(table)
Out[92]: 'Python is u gryuty progrumming lunguugy. I liky it!'
```

(20) 日期计算

```
def staytime(etc_info):
    # 计算停留时间
    # 2016-06-11#21:48:50 时间格式
    in_time = datetime.datetime.strptime(etc_info[1], "%Y-%m-
%d#%H:%M:%S")
    out_time = datetime.datetime.strptime(etc_info[2].strip(), "%Y-%m-
%d#%H:%M:%S")
    delta = (out_time - in_time).seconds
```

(21) 方差

```
var = sum([(num - ave)**2 for num in nums]) / len(nums)
```

苏大python习题

1. 给定整数m和n，如果m和n都大于1，则判定m和n是否互质，并返回判定结果。

	相关说明
输入条件	输入参数m和n是整数。大小关系未知。
输出要求	如果m和n中任何一个小于或等于1，则返回None，否则判定两数是否互质。 如果m和n互质，则返回布尔值True，否则返回布尔值False。
其他要求	将代码写入函数func1

测试用例：

输入	返回
2,3	True
4,8	False

```
# -*- coding: utf-8 -*-

def func1():
    m = eval(input("输入m:\n"))
    n = eval(input("输入n:\n"))
    if (m <= 1 or n <= 1):
        return None
    if m < n:
        m, n = n, m
    return m % n
print(bool(func1()))
```

2. 一个整数列表L=[a1, a2, ..., an]中，如果一对数(ai, aj)满足 ai>aj 且 i<j，那么这对数就称为一个逆序，列表L中逆序的数量称为逆序数。求一个整数列表L的 逆序数。

相关说明	
输入条件	列表中的元素都是整数
输出要求	如果L为空 或者 L中只有一个元素，返回0，否则返回L的逆序数。
其它要求	将代码写入函数func2

测试用例：

输入	返回
[4,3,2,1]	6
[1,3,2,4]	1

```
# -*- coding: utf-8 -*-
def fun2():
    li = list(eval(input("输入列表: ")))
    Len = len(li)
    cnt = 0
    for i in range(0, Len):
        for j in range(i, Len):
            if (li[i] > li[j]):
                cnt = cnt + 1
    print(cnt)
fun2()
```

3. 矩阵相乘：

- **输入两个整数类型的 矩阵mat1（m行d列）和 mat2（d行n列）.**
- **返回 矩阵相乘后 的结果 mat1 * mat2（m行n列）。矩阵 均用二维列表进行表示。**

	相关说明
输入条件	两个矩阵分别严格满足 m*d 和 d*n 的形状（m>=1, d>=1, n>=1，具体数值需要根据输入确定），矩阵中的元素均为整数。
输出要求	返回相乘后的矩阵，用二维列表表示，每一个元素均为整数
其他要求	将代码写入函数 func3

测试用例：

输入	返回
[[1,2]] [[1],[2]]	[[5]]
[[1,2],[1,3]] [[1,1],[1,0]]	[[3,1],[4,1]]

代码：

```
# -*- coding: utf-8 -*-


def matrixMul(A, B):
    if len(A[0]) == len(B):  # A列数=B行数
        res = [[0] * len(B[0]) for i in range(len(A))]  # 生成 A行 x B列的 0矩阵
        for i in range(len(A)):
            for j in range(len(B[0])):
                for k in range(len(B)):                    # (A列)B行
                    res[i][j] += A[i][k] * B[k][j]        # 矩阵乘法
        return res
    return ("输入矩阵有误!")

def fun3():
    mat1 = eval(input("输入矩阵mat1:"))
    mat2 = eval(input("输入矩阵mat2:"))
    result = matrixMul(mat1, mat2)
    # 矩阵乘法 结果
    print(result)
```

```
fun3()
```

4. 一维列表转成二维列表： 输入一个长度为 n*n 的一维列表， 返回一个n行n列的二维列表。

相关说明	
输入条件	一维列表能保证长度是 **n*n** (n>=1，具体数值需要根据输入确定)，且每个元素为整型。
输出要求	转换后的二维列表
其它要求	将代码写入函数 func4

测试用例：

输入	**返回**
[1]	[[1]]
[2,1,3,4]	[[2,1],[3,4]]

简易代码：

```
import numpy as np

np.reshape([1,2,3,4,5,6,7,8,9], [3, 3])

Out[88]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

详细实现：

```
# -*- coding: utf-8 -*-
import math
import numpy as np

def fun4():
    # 输入数据，例如: [2,1,4,5]
    li = eval(input("输入一个列表:"))
    print("list1:", li)

    cnt = len(li)
    print("长度数目n*n: ", cnt)

    n = int(math.sqrt(cnt))    # n x n

    # 生成 nxn 的 空矩阵
    res = [[] * n for i in range(0, n)]

    for i in range(0, n):
        for j in range(0, n):
            # 添加第i行,j列元素
            res[i].append(li[i * n + j])

    return res

res = fun4()
print(res)
```

5. 统计字符串

- **给定一个字符串，包含了若干个以 空格分开 的单词.**
- **统计 其中每个单词出现的次数，以 列表的形式 返回其中 出现 次数最多的三个单词。**
- **三者按照出现次数 降序排序，当出现次数相同时，对 单词 按照字典序降序排序。**
- **如果不足三个单词，则按照上述规则排序后全部返回。**

相关说明	
输入条件	一个只包括西文字符的字符串。
输出要求	返回一个元素是字符串的列表
其它要求	将代码写入函数func5

测试用例：

输入	返回
'hello hi hello apple'	['hello', 'hi', 'apple']
'a'	['a']

代码：

```
#-*- coding: utf8 -*-

import re
import collections

# 1. 统计单词出现次数
# 2. 以列表的形式 返回其中 出现次数最多的三个单词
# 3. 三者按照出现次数 降序排序。【次数相同，对单词按照字典序降序排序】

# 统计单词出现次数
def train(features):
    # 生成值为1的空字典
    model = collections.defaultdict(int)
    for f in features:
        model[f] += 1
    return model


def fun5():
    text = input("输入字符串: ")  # 例: i love python
    print(text)
    freq = train(text.split())

    # 遍历 freq
    print("\n打印字典: ")
    for key, value in freq.items():
        print(key, value)

    print("\n打印排序后结果:")
    # 按value排序(降序)，如果value相同则按key排序(字典序升序)
    freq = sorted(freq.items(), key=lambda x:(x[1], x[0]), reverse=True)
    print(dict(freq))
```

```
    # 将单词添加到列表
    res = []
    for key in freq[:3]:
        res.append(key[0])

    # 输出 自动
    print(res)

def main():
    fun5()

if __name__ == "__main__":
    fun5()
```

输出：

```
输入字符串: hello hi hello apple
hello hi hello apple

打印字典:
hello 2
hi 1
apple 1

打印排序后结果:
{'hello': 2, 'hi': 1, 'apple': 1}
['hello', 'hi', 'apple']
```

或者代码：

```
def func5(txt):
    counts = {}
    ls = txt.split()
    for word in ls:
        counts[word] = counts.get(word,0)+1
    items = list(counts.items())
    items.sort(key=lambda x:(x[1],x[0]),reverse=True)
    lst = []
    for i in range(len(items)):
        lst.append(items[i][0])
        if i >= 2:
            break
    return lst
```

6. Jaccard系数

- **仅包含 小写字母 的两个单词 S 和 T 的 Jaccard系数（记为J）由如下三个统计量来确定：**
- **a：在两个单词中，都出现的字母的个数**
- **b：在S中出现，但没有在T中出现的字母的个数**
- **c：在T中出现但没有在S中出现的字母的个数**
- **那么 J = a / (a + b + c)。给定两个单词S和T，求确定其Jaccard系数的三个统计量a,b,c。**

相关说明	
输入条件	两个仅包含小写字母的单词
输出要求	以元组形式返回三个统计量，即(a,b,c)
其它要求	将代码写入函数func6

测试用例：

输入	返回
‘his’, ‘she’	(2,1,1)
‘hello, ‘python’	(2,2,4)

代码：

```
#-*- coding: utf8 -*-
```

```
import re
# 仅包含 小写字母 的两个单词 S 和 T 的 Jaccard系数（记为J）由如下三个统计量来确定：
# a：在两个单词中，都出现的字母的个数
# b：在S中出现，但没有在T中出现的字母的个数      len(S) - a
# c：在T中出现，但没有在S中出现的字母的个数      len(T) - a
# 那么J = a / (a + b + c)。给定两个单词S和T
# 求确定其Jaccard系数的三个统计量a, b, c。

def fun6():
    S = input("输入单词S: ")
    T = input("输入单词T: ")
    # 分离字符 以及 生成集合[按字典序排序]
    sli = re.findall('[a-z]', S)
    slist = list(set(sli))
    tli = re.findall('[a-z]', T)
    tlist = list(set(tli))

    a = 0
    for i in range(0, len(slist)):
        for j in range(i, len(tlist)):
            if slist[i] == tlist[j]:
                a = a + 1

    b = len(slist) - a
    c = len(tlist) - a
    print((a, b, c))

def main():
    fun6()

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

7. 统计最多的字符次数

- **统计一个 非空字符串 中 出现 次数最多 的字符 及其 出现次数。**
- **其中英语字母不区分大小写，全部统计为大写字母， 如 'a' 和 'A' 在计数时进行合并为'A'。**
- **结果 以 包含字符 和 对应次数 的列表形式 进行返回。**
- **数据中 不存在 并列最多的情况，该情况不需要考虑。**

相关说明	
输入条件	能保证目标字符串非空、且其中不存在出现次数并列最多的字符
输出要求	结果以包含字符和对应次数的列表形式进行返回。
其它要求	将代码写入函数 func7

测试用例：

输入	返回
'1aA'	['A',2]
'a'	['A',1]

代码：

```
# -*- coding: utf-8 -*-
import re

# 统计一个非空字符串中
# 1. 出现 次数最多的字符 及其 出现次数
# 2. 全部统计为大写字母
# 3. 结果 [字符, 对应次数]
def func7():
    s = input("输入一个字符串: ")
    # 字母转大写
    sli = list(s.upper())
    print(sli)

    # 统计列表元素的 频率
    sli = [(i, sli.count(i)) for i in sli]

    # 去重后，排序，返回列表
    sort_sli = sorted(set(sli), key=lambda x:-x[1])
    sort_sli = [list(i) for i in sort_sli]

    print("\n排序后去重的列表: ", sort_sli)

    print("\n次数最多: ", sort_sli[0])
```

```
    # 转换成字典: {'A': 5, 'D': 2, '1': 1, 'B': 1, 'G': 1}
    print(dict(sort_sli))

def main():
    func7()

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

输出：

```
输入一个字符串: 1231AAAsdbafg
['1', '2', '3', '1', 'A', 'A', 'A', 'S', 'D', 'B', 'A', 'F', 'G']

排序后去重的列表:  [['A', 4], ['1', 2], ['B', 1], ['2', 1], ['F', 1], ['D', 1], ['3', 1], ['S', 1], ['G', 1]]

次数最多:  ['A', 4]
{'A': 4, '1': 2, 'B': 1, '2': 1, 'F': 1, 'D': 1, '3': 1, 'S': 1, 'G': 1}
```

8. 数字和平均值

- **一个字符串中存在多个正整数.**
- **请提取出 位数在[3,5] 之间的所有正整数，构成一个列表**
- **对此列表按照 数字和平均值 （各位数字的总和/位数）进行 降序排序， 并返回排序结果列表。**
- **数字和平均值 就是 各位数字的总和除以位数，例如2345的数字和平均值=(2+3+4+5)/4=3.5，12的数字和平均值=(1+2)/2=1.5。**

相关说明	
输入条件	存在多个正整数的字符串
输出要求	结果以满足要求的列表形式进行返回。如原字符串中不存在满足条件的正整数，返回None
其它要求	将代码写入函数func8

测试用例：

输入	返回
'123a4567 1'	[4567,123]
'1234'	[1234]

代码：

```
# -*- coding: utf-8 -*-

import re

def sumAve(x):
    tmp = x
    cnt = 0
    res = 0
    while tmp:
        res += tmp%10
        tmp = int(tmp / 10)
        cnt = cnt + 1
    return res / cnt

def func8():
    num = input("输入字符串:")
    # 注意 [0-9]{3,5} 不要乱加空格，否则出错
    li = re.findall('[0-9]+', num)
    print(li)
    li = [int(elem) for elem in li if len(elem) >= 3 and len(elem) <= 5]
    print(li)

#    print(sumAve(2345))
    # 按照 数字和平均值 排序
    li.sort(key=lambda x:sumAve(x), reverse=True)
    print(li)

def main():
    func8()

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

输出：

```
输入字符串:123asd235 43 sdg535 35 345
[123, 235, 535, 345]
[535, 345, 235, 123]
```

苏大真题[2005-2019]

2005. 把一个数表示成若干个素数的和.

```
# -*- coding: utf-8 -*-

import math

def is_prime(num):
    if num < 2:
        return False

    top = int(math.sqrt(num))
    i = 2
    while i <= top:
        if num % i == 0:
            return False
        i = i + 1

    return True

res = []
# 从大到小 继续筛选
def split_prime(num):
    if num < 2:
        return

    if is_prime(num):
#         print(num)
        res.append(num)
        return

    for i in range(num, 1, -1):
        if (is_prime(i) and num - i > 1):
```

```
            res.append(i)
            split_prime(num - i)
            return

def faction_prime():
    while True:
        num = int(input())
        split_prime(num)
        print(res)
        res.clear()

def main():
    faction_prime()

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

输出：

```
4
[2, 2]
6
[3, 3]
7
[7]
22
[19, 3]
45
[43, 2]
```

或者：

```
0 not in [n%d for d in range(2, n)]
```

2005.2 统计篇文章中各英文字母的个数，并排序.

2005_2.txt

```
Mr. Sherlock Holmes, who was usually very late in the mornings,
sort which is known as a "Penang lawyer." Just under the head was a
```

代码：

```
# -*- coding: utf-8 -*-

import re

def words(text):
    return re.findall('[a-z]', text.lower())

def train(features):
    model = {}
#    print(features)
    for f in features:
        model[f] = model.get(f, 0) + 1

    return model

def main():
#    t = open("./file/2005_2.txt").read()
#    print(t)
#    print(words(t))
    statistic = train(words(open("./file/2005_2.txt").read()))
    statistic = list(statistic.items())
    print(statistic)
    statistic.sort(key=lambda x:(x[1], x[0]), reverse=True)
    print(dict(statistic))

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

输出：

```
[('m', 3), ('r', 7), ('s', 10), ('h', 8), ('e', 10), ('l', 6), ('o', 6),
('c', 2), ('k', 2), ('w', 6), ('a', 10), ('u', 4), ('y', 3), ('v', 1),
('t', 5), ('i', 4), ('n', 8), ('g', 2), ('p', 1), ('j', 1), ('d', 2)]
{'s': 10, 'e': 10, 'a': 10, 'n': 8, 'h': 8, 'r': 7, 'w': 6, 'o': 6, 'l':
6, 't': 5, 'u': 4, 'i': 4, 'y': 3, 'm': 3, 'k': 2, 'g': 2, 'd': 2, 'c':
2, 'v': 1, 'p': 1, 'j': 1}
```

2006. 找出 100 到 1000 内的不含 9 的素数，存到 result 文件中.

```
# -*- coding: utf-8 -*-

import math

def is_prime(num):
    if num < 2:
        return False
    top = math.sqrt(num)
    i = 2

    while i <= top:
        if num % i == 0:
            return False
        i = i + 1

    return True

# 不含9
def judge(num):
    a = list(str(num))
    return '9' not in a

def filterPrime(start, end):
    return [i for i in range(start, end + 1) if (is_prime(i) and
judge(i))]

def save2file(text):
    print(text)

    with open('./file/2006.txt', 'w+', encoding='utf8') as f:
        f.write(text)

def main():
    res = []
    res = filterPrime(100, 1000)
    save2file(str(res))

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

2006.txt

```
[101, 103, 107, 113, 127, 131, 137, 151, 157, 163, 167, 173, 181, 211,
223, 227, 233, 241, 251, 257, 263, 271, 277, 281, 283, 307, 311, 313,
317, 331, 337, 347, 353, 367, 373, 383, 401, 421, 431, 433, 443, 457,
461, 463, 467, 487, 503, 521, 523, 541, 547, 557, 563, 571, 577, 587,
601, 607, 613, 617, 631, 641, 643, 647, 653, 661, 673, 677, 683, 701,
727, 733, 743, 751, 757, 761, 773, 787, 811, 821, 823, 827, 853, 857,
863, 877, 881, 883, 887]
```

2007. 素数

- **把 10 到 1000 之间满足以下两个条件的数，存到 result.txt 文件中.**
- **是素数.**
- **它的反数也是素数，如： 123 的反数是 321 .**

```
# -*- coding: utf-8 -*-

# 把10 到 1000 之间满足以下两个条件的数，存搭配result.txt文件中
# 1. 是素数
# 2. 它的反数也是素数，如：123的反数是321
import math

def is_prime(num):
    if num < 2:
        return False
    top = math.sqrt(num)
    i = 2

    while i <= top:
        if num % i == 0:
            return False
        i = i + 1
    return True

# 123 的 反数是 321
def reverseNum(num):
    res = int(str(num)[::-1])
    return res

# 筛选素数
def filter_prime(start, end):
    return [i for i in range(start, end + 1) if is_prime(i) and
```

```
is_prime(reverseNum(i))]

def saveFile(text):
    with open('./file/2007.txt', 'w', encoding='utf8') as f:
        f.write(str(text))

def main():
    res = filter_prime(10, 1000)
    print(res)
    saveFile(res)

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

2007.txt

```
[11, 13, 17, 31, 37, 71, 73, 79, 97, 101, 107, 113, 131, 149, 151, 157,
167, 179, 181, 191, 199, 311, 313, 337, 347, 353, 359, 373, 383, 389,
701, 709, 727, 733, 739, 743, 751, 757, 761, 769, 787, 797, 907, 919,
929, 937, 941, 953, 967, 971, 983, 991]
```

2008. 文件读取字符串，排除The

- **用 IE 从 FTP 上下载 org.dat ，并保存在 D 盘的根目录中.**
- **此文件中按文本方式存放了一段其他文章，其中有若干长度小于 15 的英文单词，单词之间用空格分开，无其他符号.**
- **顺序读取这段文章的不同的单词(大小写敏感)，同时在 读取的过程中 排除所有的单词 THE 以及变形，即这些单词不能出现在读取的结果中.**
- **将读取的所有单词的首字母转大写后，输出 D 根目录下 new.txt ，每个单词一行.**

```
# -*- coding: utf-8 -*-

import re

def words(text : str):
    return re.findall('[a-zA-Z]+', text)
#    return text.split()                    # 只匹配字符串

def train():
    with open("./file/2008.dat", 'r', encoding='utf8') as f:
        wds = f.read()
```

```
    wds = words(wds)
    print(wds)

    wds = [i.capitalize() for i in wds if i.lower() != "the"]
    print("\n", wds)

    with open("./file/2008_new.txt", 'w', encoding='utf8') as f:
        for word in wds:
            f.write(word+'\n')

train()
```

2008.dat

```
The constructor is used to initialize the object The destructor is used
to delete the Object the
calling seqence of constructor is opposite to the calling sequence of
destructor
```

2008_new.txt

```
Constructor
Is
Used
To
略
```

2009. 文件读取字符串，转数字【八进制/十进制】

- 用 IE 浏览器从 FTP 上下载 org.dat ，并保存在 D 盘的根目录下.
- 此文件中按文本方式存放了一段其他文章，其中有 若干长度小于 15 的 **十进制** 或 **八进制** 数字，数字之间用 **,** 分开，数字内部 存在且 仅存在 **空格**.
- 八进制数 **以起始位 0** 作为标示 与 十进制数区分.
- 顺序读取这些数字 将他们 **转变为十进制数** 后按 **从大到小** 的顺序排序后，输出到 D 盘根目录下 new.txt ，每个数字一行. eg ： *235*,34_*2*,043_1_,1_3 ，分别是：十进制 235 ，十进制 342 ，八进制 431 ，十进制 13 ， _ 代表 空格.

```
# -*- coding: utf-8 -*-

def int10(x):
```

```
    if x[0] == '0':
        return int(x, 8)
    return int(x)

def main():
    with open("./file/org.dat", 'r+', encoding='utf8') as f:
        wds = f.read()

    # 按照","分割字符串
    words = wds.split(',')
    # 去除空格，数字字符串转化成10进制整数
    words = [i.replace(" ","") for i in words]
    print(words)

    trans = list(map(int10, words))
    trans.sort(reverse=True)
    print(trans)

    with open('./file/2009_new.txt', 'w', encoding='utf8') as f:
        f.write(str(trans))

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

org.dat

```
 235 ,34  2, 043 1 ,1 3
```

2009_new.txt

```
[342, 281, 235, 13]
```

2010. 读取二进制文件 以及筛选素数

- 从 FTP 上下载 make.exe 和 org.dat ，运行 make.exe 输入准考证后三位
- 生成 data.txt 文件为二进制编码 **data.txt** ，内存有 2048 个整数.
- 其中 前 **n** 个为非 0 数，后 **2048 - n** 个数为 0 ，将其读入数组，计算 **非零数** 的 **个数** n.
- 选出 n 个数中 的 **最大数** 和 **最小数**，选出 n 个数中 **最大素数**.
- 将 n 个数 **从大到小排序**，并 **平均分成三段** (若 n 非 3 的整数倍，则不考虑最后的 1-2 个数)，
- 选出中间段的最大数和最小数.

数据:

```
0800 0000 0600 0000 0200 0000 1900 0000
0500 0000 0200 0000 0300 0000 0900 0000
0100 0000 0d0a 0000 0022 0000 000c 0000
002d 0000 0053 0000 0066 0000 0007 0000
0005 0000 000d 0000 004f 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
00
```

输出 :

```
26112 1
数据数: 16 [26112, 21248, 11520, 8704, 3072, 2573, 1792, 25, 9, 8, 6, 5,
3, 2, 2, 1]
最大素数:  5
2573 8
```

代码 :

```
# -*- coding: utf-8 -*-
import math
import random
import struct

def is_prime(num):
    if num < 2:
        return False
    top = int(math.sqrt(num))
    i = 2
    while i <= top:
        if num % i == 0:
            return False
        i = i + 1
    return True

def main():
    res = [random.randrange(0, 9999, 1) for i in range(0, 16)]
#    print(res)
```

```
#    with open('./file/2010.txt', 'rb+') as f:
#        for i in res:
#            s = struct.pack('i', i)
#            f.write(s)
    len1 = len(res)
    nums = []
    with open('./file/2010.txt', 'rb+') as f:
        for i in range(len1):
            data = f.read(4)
            elem = struct.unpack('i', data)[0]
            nums.append(elem)

    # 按照 16进制 转换成 10进制
    nums.sort(reverse=True)
    print(nums[0], nums[-1])

    print("数据数:", len1, nums)
    for i in nums:
        if (is_prime(i)):
            print("最大素数: ", i)
            break

    mid = nums[len1//3 : len1//3*2]
    print(mid[0], mid[-1])

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

2011. 筛选素数

- 输出 1000-9999 中满足以下条件的所有数：
- 该数是素数.
- **十位数** 和 **个位数** 组成的数是 **素数**， **百位数** 和 **个位数** 组成的数是 **素数**.
- **个位数** 和 **百位数** 组成的数是 **素数**，**个位数** 和 **十位数** 组成的数是 **素数。** 比如 1991 ，**个位** 和 **十位** 组成的数就是 19 .

```
# -*- coding: utf-8 -*-

import math
```

```
def is_prime(num):
    if num < 2:
        return False
    top = int(math.sqrt(num))
    i = 2
    while i <= top:
        if num % i == 0:
            return False
        i = i + 1
    return True

def judge(num):
    unit = num % 10             # 个位
    decade = (num // 10) % 10   # 十位
    hund = (num // 100) % 10    # 百位
    return [unit, decade, hund]

def comb(a, b):
    return a * 10 + b

def filter_prime(start, end):
    res = [i for i in range(start, end + 1) if (is_prime(i)) and
           (is_prime(comb(judge(i)[1], judge(i)[0]))) and
           (is_prime(comb(judge(i)[2], judge(i)[0]))) and
           (is_prime(comb(judge(i)[0], judge(i)[1]))) and
           (is_prime(comb(judge(i)[0], judge(i)[2]))) ]
    return res

def main():
    res = filter_prime(1000, 9999)
    print(res)

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

输出：

```
[1117, 1171, 1997, 2111, 2113, 2131, 2137, 2311, 2371, 2711, 2713, 2731,
2917, 3137, 3331, 3371, 3779, 3917, 4111, 4337, 4397, 4937, 5113, 5171,
5197, 5711, 5779, 6113, 6131, 6173, 6197, 6311, 6317, 6337, 6397, 6779,
6917, 6997, 7331, 7937, 8111, 8117, 8171, 8311, 8317, 8713, 8731, 8779,
9137, 9173, 9311, 9337, 9371, 9397]
```

2012. 读取二进制文件 以及 左标

- 从服务器上下载数据文件 org.dat 文件以 **二进制方式** 存放一系列整数，每个整数占 **4 个字节**.
- 从 第一个整数 开始，**第一个整数** 和 **第二个整数** 构成一个坐标点，以此类推，**数据文件** 中保存了 许多 **坐标点数据**.
- 规定处于 **第一象限的坐标点** 为 **有效点**，请问 数据文件 中 **所有点的个数 n** 为多少？ **有效点的个数 k** 为多少？
- 每个 **有效点** 与 **坐标原点** 构成一个的 **矩形**，请问 **k 个有效点** 与 **坐标原点** 构成的 **k 个矩形** 的 最小公共区域面积为多少？
- 寻找 **有效点** 中 符合下列条件的点：
  - 以该点为 **坐标原点**， 其它 **有效点** ：
    - 仍然是 **有效点**， 即 **处于第一象限** (不包括坐标轴上的点).
    - 输出这些点，对 所有有效点 进行分组，每个有效点 有且只有 属于一个分组，分组内的点 符合下列规则：
      - 若 对组内所有点的 x，坐标进行排序。
      - 点 **p1(x1, y1)** 在点 **p2(x2, y2)** 后面，即 x1 > x2 那么 y1 > y2 ，请输出所有的分组。

```
# -*- coding: utf-8 -*-

import struct

# 读取未知数字个数的二进制文件
def readFile(url):
    ans = []
    with open(url, 'rb+') as f:
        while True:
            data = f.read(4)
            if not data:
                break
```

```
            elem = struct.unpack('i', data)[0]
            ans.append(elem)

        return ans

# 将读取到的数构成坐标
def point(data):
    res = []
    len_data = len(data)
    for i in range(0, len_data, 2):
        comb = [data[i], data[i + 1]]
        res.append(comb)
    return res

# 有效点的个数
def validPoint(point):
    res = [elem for elem in point if (elem[0] > 0 and elem[1] > 0)]
    return res

# 横坐标的最小值 和 纵坐标的最小值 组成的矩阵
def minSquare(point):
    px, py = point[0][0], point[0][1]
    len_p = len(point)
    for i in range(len_p):
        if px > point[i][0]:
            px = point[i][0]
        if py > point[i][1]:
            py = point[i][1]
    return px * py

# 横坐标最小的同时纵坐标也是最小(取最小值的下标是否一致)，返回满足条件的点
def minPoint(point):
    minx, miny = 0, 0
    len_p = len(point)
    for i in range(len_p):
        if point[minx][0] > point[i][0]:
            minx = i
        if point[miny][0] > point[i][1]:
            miny = i
    if minx == miny:
        return point[minx]
    else:
```

```
        return '不存在这样的点'

# 分组
def groupX(point : list):
    # 先按坐标x轴升序排列
    point.sort(key=lambda x : x[0])
    print(point)

    count = 0
    flag = [0 for i in range(len(point))]  # 标记数组初始化为0
    p_len = len(point)
    for i in range(p_len):
        if flag[i] == 0:                   # 首次被标记，归为下一个分组
            count += 1
            # 组数:count
            print('\ncount = ', count)
            print(point[i], end='')
            flag[i] = 1                    # 标志着已经使用过
            # t 为 第 t 组
            t = i
        for j in range(i + 1, p_len):
            if (flag[j] == 0) and (point[j][1] > point[t][1]) and
(point[j][0] > point[t][0]):
                print(point[j], end='')
                flag[j] = 1
                t = j

def main():
    data = readFile("./file/file_2012.txt")
#    print(data, "\n")

    p_point = point(data)
#    print(p_point)

    # 有效点的个数
    vapoint = validPoint(p_point)
    print(vapoint)
    print("有效点的个数 = {0}\n".format(len(vapoint)))
    print("最小公共区域面积 = {0}".format(minSquare(vapoint)))
    print("符合条件的点: ", minPoint(vapoint))
    groupX(vapoint)
```

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

输出：

```
[[20, 9], [1, 1], [8, 9], [90, 2], [6, 8], [80, 3], [20, 3], [4, 3], [22,
77], [90, 10], [8, 6], [8, 90], [77, 99], [8, 9]]
有效点的个数 = 14

最小公共区域面积 = 1
符合条件的点:  [1, 1]
[[1, 1], [4, 3], [6, 8], [8, 9], [8, 6], [8, 90], [8, 9], [20, 9], [20,
3], [22, 77], [77, 99], [80, 3], [90, 2], [90, 10]]

count =  1
[1, 1][4, 3][6, 8][8, 9][22, 77][77, 99]
count =  2
[8, 6][20, 9][90, 10]
count =  3
[8, 90]
count =  4
[8, 9]
count =  5
[20, 3]
count =  6
[80, 3]
count =  7
[90, 2]
```

2013. 图论：两顶点间存在长度为k的路径

PathInput.txt

```
6
[PVG, CAN]
[CAN, PEK]
[PVG, CTU]
[CTU, DLC]
[DLC, HAK]
[HAK, LXA]
```

PathRequest.txt

```
2
[PVG, DLC, 2]
[PVG, LXA, 2]
```

Output.txt

```
[PVG, DLC, YES]
[PVG, LXA, NO]
```

代码：

```
# -*- coding: utf-8 -*-
# 1. PVG -> PEK
# 即：PVG -> CAN -> PEK
# PathInput.txt
# 6
# [PVG, CAN]
# [CAN, PEK]
# [PVG, CTU]
# [CTU, DLC]
# [DLC, HAK]
# [HAK, LXA]
# PathRequest.txt
# 2
# [PVG, DLC, 2]
# [PVG, LXA, 2]
import re

# 读文本文件内容，以换行分隔单词，返回字符串列表
def readFile(url):
    with open(url, 'r+', encoding='utf8') as f:
        wds = f.readlines()
        return wds

# 两顶点间 存在 长度为k的路径
# countLine: Line的长度
# 递归判断起点为: start
# 终点为: end
# 长度为k的路径是否存在
```

```
def check(Line, countLine, start, end, k):
    flag = 0
    # 遍历所有 Line 的数据
    # [['PVG', 'CAN'], ['CAN', 'PEK'], ['PVG', 'CTU'], ['CTU', 'DLC'],
['DLC', 'HAK'], ['HAK', 'LXA']]
    for i in range(countLine):
        # 找到路径
        if k == 0 and Line[i][0] == start and Line[i][1] == end:
            flag = 1
            break
        elif k > 0 and Line[i][0] == start and Line[i][1] != end:
            k -= 1
            # 改变 起点
            flag = check(Line, countLine, Line[i][1], end, k)

    return flag

# Line: Line的数据
# Path: Path的数据
def writeFile(url, Line, countLine, Path, countPath):
    with open(url, 'w', encoding='utf8') as f:
        # 操作次数
        for i in range(countPath):
            # 起点, 终点, step=2
            # 两顶点间 存在 长度为k的路径
            if check(Line, countLine, Path[i][0], Path[i][1], Path[i]
[2]):
                f.write('[{0}, {1}, Yes]\n'.format(Path[i][0], Path[i]
[1]))
            else:
                f.write('[{0}, {1}, No]\n'.format(Path[i][0], Path[i]
[1]))

def main():
    dataLine = readFile('./file/file_2013_line.txt')
    dataPath = readFile('./file/file_2013_plan.txt')
    countLine, countPath = int(dataLine[0]), int(dataPath[0])
#    print(countLine, countPath)

    Line, Path = [], []
    for i in range(1, countLine + 1):
        Line.append([dataLine[i][1:4], dataLine[i][6:9]])
```

```
#         print(dataLine[i])

    for i in range(1, countPath + 1):
        Path.append([dataPath[i][1:4], dataPath[i][6:9], int(dataPath[i]
[11])])
#         print(dataPath[i])

    # 输出 Line 的数据
    print(dataLine, "\n")
    print(Line, "\n")
    # 输出 Path 的数据
    print(dataPath, "\n")
    print(Path)

    writeFile('./file/file_2013_output.txt', Line, countLine, Path,
countPath)
if __name__=='__main__':
    main()
```

2014. 坐标距离

- 从网页上下载 input.dat 文件，里面是用二进制编写的，里面放了一堆 int 型的数，
- 每个数占 4 个字节，每次读取两个，这两个数构成一个坐标.
- 规定处于第一象限的数是有效点(即 x>0, y>0 的坐标)，问这么多点中有效点有多少个？
- 现在用户从键盘输入一个坐标和一个数字 k ，设计算法输出 k 个离该坐标距离最近的点的坐标和每个坐标到

  该点的距离，写入到 output.txt 文件中

```
# -*- coding: utf-8 -*-
import struct
import math

def readFile(url):
    res = []
    with open(url, 'rb+') as f:
        while True:
            data = f.read(8)
            if not data:
                break
            elem = struct.unpack('2i', data)
```

```
            res.append(elem)
    return res

def validPoint(data):
    res = [elem for elem in data if (elem[0] > 0 and elem[1] > 0)]
    return res

def distance(a, target):
    return math.pow(abs(a[0]-target[0]), 2) + math.pow(abs(a[1]-
target[1]), 2)

def main():
    data = readFile('./file/file_2014.txt')
    print(data)

    vapoint = validPoint(data)
    print("\n有效点: ", vapoint)
    print("有效点数目: ", len(vapoint))

    k = int(input("输入k:"))
    x = int(input("输入坐标x:"))
    y = int(input("输入坐标y:"))
    target = [x, y]
    print(target)

    sorted_vap = sorted(vapoint, key=lambda x:distance(x, target))
    print(sorted_vap)

    print("k个坐标距离target的距离：")
    for i in range(0, k):
        print("{0} -> 目标{1} 距离: {2:.2f}".format(sorted_vap[i], target,
math.sqrt(distance(sorted_vap[i], target)) ))


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

2015. 二进制坐标距离

- 从网页上下载 input.dat 文件，里面是用二进制编写的，里面放了一堆 int 型的数，每个数占 4 个字节，每 次读取两个，这两个数构成一个坐标.

- 规定处于第一象限的数是有效点(即 x>0, y>0 的坐标)，问这么多点中有效点有多少个？
- 从键盘上输入 k 和 n ，从第一问中的有效点中找出距离小于 n ，距离小于 n 的点的个数要大于 k ，将它们 以文本格式输出到文件中.

```
# -*- coding: utf-8 -*-

import struct
import math

def readFile(url):
    res = []
    with open(url, 'rb+') as f:
        while True:
            data = f.read(8)
            if not data:
                break
            elem = struct.unpack('2i', data)
            res.append(elem)
    return res

def validPoint(points):
    res = [elem for elem in points if (elem[0] > 0 and elem[1] > 0)]
    return res

def distance(a, target):
    return math.pow(abs(a[0] - target[0]), 2) + math.pow(abs(a[1] -
target[1]), 2)


def writeFile(sorted_vap, target, k):
    with open("./file/file_2015_output.txt", 'w', encoding='utf8') as f:
        for i in range(0, k):
             f.write("{0} -> 目标{1} 距离:
{2:.2f}\n".format(sorted_vap[i], target,
math.sqrt(distance(sorted_vap[i], target))))


def main():
    points = readFile("./file/file_2015.txt")
    vapoints = validPoint(points)
    print(vapoints)
```

```
    valen = len(vapoints)
    print("有效点的个数: ", valen)

    k = int(input("输入k:"))
    n = int(input("输入n:"))
    x = int(input("输入坐标x:"))
    y = int(input("输入坐标y:"))
    target = [x, y]

    sorted_vap = sorted(vapoints, key=lambda x:(distance(x, target)))
    print("k个坐标距离target的距离：")
    for i in range(0, valen):
        dis = math.sqrt(distance(sorted_vap[i], target))
        if dis < n:
            print("{0} -> 目标{1} 距离: {2:.2f} 小于
{3}".format(sorted_vap[i], target,  dis, n))

    writeFile(sorted_vap, target, valen)


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

输出：

```
[(20, 9), (1, 1), (8, 9), (90, 2), (6, 8), (80, 3), (20, 3), (4, 3), (22,
77), (90, 10), (8, 6), (8, 90), (77, 99), (8, 9)]
有效点的个数:  14

输入k:3

输入n:16

输入坐标x:4

输入坐标y:4
k个坐标距离target的距离：
(4, 3) -> 目标[4, 4] 距离: 1.00 小于16
(1, 1) -> 目标[4, 4] 距离: 4.24 小于16
(6, 8) -> 目标[4, 4] 距离: 4.47 小于16
(8, 6) -> 目标[4, 4] 距离: 4.47 小于16
```

```
(8, 9) -> 目标[4, 4] 距离: 6.40 小于16
(8, 9) -> 目标[4, 4] 距离: 6.40 小于16
```

2016. 词频统计

file_2016_in.txt

```
Mr. Sherlock Holmes, who was usually very late,
略
```

file_2016_out.txt

```
the, 15
night, 9
略
```

代码：

```
# -*- coding: utf-8 -*-
# 文本文件 input.txt 由若干英文单词和分隔符(空格，回车，换行)构成.
# 根据如下说明编写程序：
# 统计不同单词出现的次数(频度).
# 将统计结果按出现频度从高到低排序，并将出现频度大于 5 的单词及其频度输出到文件
output.txt
# 中. 文件格式如图所示
# 多个连续的分隔符被视为一个分隔符.
# 大小写敏感.
# 每个单词的长度不超过 20 个字符.
import re

def readFile(url):
    with open(url, 'r+', encoding='utf8') as f:
        data = f.read()
    return data

def train(words):
    freq = {}
    for word in words:
        freq[word] = freq.get(word, 0) + 1

    return freq
```

```
def writeFile(sorted_freq):
    with open('./file/file_2016_output.txt', 'w', encoding='utf8') as f:
        for elem in sorted_freq:
            if elem[1] > 5:
                f.write("{0}, {1}\n".format(elem[0], elem[1]))


def main():
    data = readFile('./file/file_2016_in.txt')
    words = re.findall('[a-zA-Z]+', data)
    print(words)

    freq = train(words)
    sorted_freq = sorted(freq.items(), key=lambda x:(x[1]), reverse=True)
    print("\n", sorted_freq, "\n")
    for elem in sorted_freq:
        if elem[1] > 5:
            print("{0}, {1}".format(elem[0], elem[1]))

    writeFile(sorted_freq)


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

2017. 坐标问题

已知：二进制数据文件 data.bin 中存放了若干个整数，请编写程序完成如下功能： 编写程序读取所有数据.

- 以每相邻两个整数为一对按顺序构成二维平面上的坐标点. 例如：有数据 12 ， 34 ， 53 ， 25 ， 61 ， 28 ， 78 等，则构成六个坐标点如下： (12, 34) 、 (34, 53) ， (53, 25) , (25, 61) , (61, 28) , (28, 78) ；
- 以 每个坐标点为圆心，以 该点 与 其后面第一个点 的 欧氏距离为半径 r . 计算 **最后一个点** 时 以其 **和第一个点** 的欧氏距离为半径。
- 计算 **每个圆包含的坐标点数** 。
- 例如： 坐标点 (12, 34) 的圆半径$r = \sqrt{(12-34)^2 + (34-53)^2}$是坐标点 (12, 34) 与 (34, 53) 的欧式距离. 坐标点 (28, 78) 的圆半径$r = \sqrt{(28-12)^2 + (78-34)^2}$是坐标点 (28, 78) 与 (12, 34) 的欧式距离.

- 计算所有圆的**点密度值**，然后输出 **点密度值** 最大的 **5 个坐标点**以及 相应圆中包含的 **点数** 和 **点密度值**. 输出格式 要求：

坐标点	包含点数	点密度
(x坐标，y坐标)	(占5列，右对齐)	(占7列，右对齐，保留2位小数)

- 上述文字部分不需要显示. 其中：**圆的点密度** 为 **圆包含的点数** /**圆面积**，如果点在圆上，则也算圆包含该点，在计算点密度时，圆心也算一个点. 计算圆面积时$\pi = 3.14$. 例如：坐标点 (2, 1) ，则该坐标点也属该坐标点的圆内的一个点.

```
# -*- coding: utf-8 -*-
import struct
import math

def readFile(url):
    # 测试用例
    wds = """2 88 59 83 87 65 38 72 70 76 50 62 4 76 68 70 50 60 13 74 66
60 8 28 97 94 99 52 6 90 69 60 54
83 76 89 64 73 48 69 83 28 84 67 14 50 99 86 35 36 5 82 67 36 92 99 44 27
53 76 24 45 27 19 14
65 86 69 47 80 96 96 10 68 60 91 87 25 15 50 8 18 3 15 85 88 14 8 2 64 63
62 70 58 62 93 51 66
62 73 75 6"""
    # 写文件
    with open(url, 'wb+') as f:
        line = wds.split()
        for i in line:
            elem = struct.pack('i', int(i))
            f.write(elem)

    res = []
    # 读文件
    with open(url, 'rb+') as f:
        while True:
            data = f.read(8)
            if not data:
                break
            elem = struct.unpack('2i', data)  # 读一个元组(x,y)
            res.append(elem)
    return res

# 计算欧式距离
```

```
def distance(a, target):
    return math.sqrt(math.pow(abs(a[0] - target[0]), 2) +
math.pow(abs(a[1] - target[1]), 2))

# 计算 圆的点密度
def density(count, r):
    return count / (r*r*math.pi)

def main():
    points = readFile('./file/file_2017.txt')
    polen = len(points)

    circle_radius = []   # 存储距离(当作圆半径)
    # 以每个坐标点为圆心，以 该点 与 其后面第一个点 的欧氏距离为半径 r
    # 计算每个圆包含的坐标点数. 计算 最后一个点 以其和 第一个点 的欧氏距离为半径.
    for i in range(0, polen):
        circle_radius.append( distance(points[i], points[(i+1)%polen]) )

    # 计算 每个圆包含的 坐标点数
    count = [0 for i in range(0, polen)]   # 圆包含的点数
    for i in range(0, polen):
        for j in range(0, polen):
            # 坐标i 和 其他坐标j 距离
            if (distance(points[i], points[j]) - circle_radius[i] <= 1e-
8):
                count[i] += 1              # 圆心为i,radius[i]的圆, 包含的坐
标 数

    # 计算所有圆的点密度值，然后输出 点密度值 最大的 5 个坐标点
    # 圆的点密度：圆包含的点数 / 圆面积
    densitys = []
    for i in range(0, polen):
        densitys.append( density(count[i], circle_radius[i]) )

    # 输出 点密度值 最大的 5 个坐标点
    combine = []
    for i in range(0, polen):
        combine.append([points[i], count[i], densitys[i]])

    # 根据点密度排序
    combine.sort(key=lambda x:(x[2]), reverse=True)
    # 输出格式:
```

```
    # 坐标点(x,y)  包含点数(占5列,右对齐)  点密度(占7列，右对齐，保留2位小数)
    for i in range(0, 5):
        print("坐标点:{0}, 包含点数:{1:>5}, 点密度:
{2:>7.2f}".format(combine[i][0], combine[i][1], combine[i][2]))

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

输出：

```
坐标点:(63, 62), 包含点数:    4, 点密度:   0.02
坐标点:(8, 18), 包含点数:    2, 点密度:   0.02
坐标点:(64, 73), 包含点数:   13, 点密度:   0.02
坐标点:(68, 70), 包含点数:   18, 点密度:   0.01
坐标点:(51, 66), 包含点数:    7, 点密度:   0.01
```

2016. 保研 查字典匹配句子

- 请从服务器将两个数据文件 input.txt 和 words.txt 下载到本地电脑的 D 盘根文件夹。
- 在 D 盘根文件夹的 **words.txt** 中存储了不超过 30000 条的英文单词，**每个单词占一行。**单词的最大长度为 20 ，且单词内部没有空格，文件中无重复单词。
- 在D盘根文件夹的 **input.txt 中存储了一个「丢失」了空格和标点符号的英文文章**。每行不超过 128 个字 符，请编写程序把该文章中第一行和最后一行显示在屏幕上。
- 编写程序：**将 words.txt 中的最后三行显示在屏幕上**；
- 编写程序：**利用 words.txt 中的单词作为词典**，采用**正向最大匹配切分单词算法对 input.txt 中的文本进行单词切分**。切分时单词区分大小写，切分分割标记采用空格，并将切分后的结果写入到 out.txt 中。
- 所谓正向最大匹配切分就是从左向右扫描待切分字符串，尽量取长词。
- 下面举一个简单例子：现有待切分字符串 ABCDEFGHIJ ，设词典中最大单词长度为 5 。那么**按照算法首先取 出 ABCDE 判断是否是单词**，如果是则切分到一个单词，否则**舍弃最后一个字母接着判断**，也就是判断 ABCD 是否是单词，依此类推，当只有一个字母时可以直接认定为是单词。在成功切分出一个单词后对待切分字符 串余下的部分再次执行上述过程。
- 编写程序实现步骤 2 、 3 描述的要求，并通过如下所示的主函数对进行验证，注意：除了指定添加的代码之 外，不得修改 main 函数其余部分。对 main 函数每修改一处，总分扣 3 分，最多扣 10 分。

file_input_2016.txt

```
whenIwasyoung
ithinkth
```

file_word_2016.txt

```
apple
love
```

output

```
['apple', 'love', 'dog', 'competitive', 'judge', 'man', 'people', 'when',
'I', 'was',
dic: appl time
input: whenIwasyoung Totaltimeapple
['when', 'I', 'was', 'young', 'i', 'think', 'competitive', 'mechanism',
```

代码：

```
# -*- coding: utf-8 -*-
def readFile(url):
    with open(url, 'r+', encoding='utf8') as f:
        data = f.read()
        txt = data.split('\n')
    return txt

def splitEssay(sentence, words):

    start, sen_len = 0, len(sentence)
    res = []
    while start <= sen_len:
        for k in range(20, 0, -1):
            if sentence[start:start+k] in words:
                res.append(sentence[start:start+k])
                break
        start += k
    return res

def main():
    words = readFile('./file/file_word_2016.txt')
    print("词典: ", words, "\n")
    essay = readFile('./file/file_input_2016.txt')
```

```
    print("无空格文章: ", essay, "\n")

    for word in words[-3:]:
        print(word)

    # 显示文章的第一行 和 最后一行
    first = splitEssay(essay[0], words)
    last = splitEssay(essay[-1], words)

    print("\n文章第一行: {0}\n".format(' '.join(first)))
    print("文章最后一行: {0}\n".format(' '.join(last)))

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

2018. 真题 最大公约数

- 有20000个数存储于二进制文件中，读取出来-
- 然后求一个最大子集，其中两两互相不为倍数，不为约数，最大公约数为1
- 然后满足的数据输出到指定文件中。

```
# -*- coding: utf-8 -*-
import struct
import random
import math

def GenerateData(url):
    random.seed(100)
#     res = (random.randint(2,20000) for i in range(0, 20000))
#     res = [11, 3, 5, 7, 9, 12, 33, 31, 4, 11]
    res = [2,3,5,7,13,17,23,29,31,37,41,43, 9]
#     res = [i for i in range(1, 500)]
    with open(url, 'wb+') as f:
        for elem in res:
            s = struct.pack('i', elem)
            f.write(s)

def readFile(url):
    with open(url, 'rb+') as f:
        res = []
        while True:
```

```
            data = f.read(4)
            if not data:
                break
            elem = struct.unpack('i', data)[0]
            res.append(elem)
        return res

def writeFile(data, length):
    with open("./file/file_2018_output.txt", 'w+', encoding='utf8') as f:
        for i in range(0, length, 5):
            if data[i:i+5]:
                f.write(str(data[i:i+5]) + "\n")

def gcd(num1, num2):
    if num1 < num2:
        num1, num2 = num2, num1

    while (num1 % num2):
        temp = num1 % num2
        num1 = num2
        num2 = temp

    return num2

def filter_list(data, length):
    res = []
    print(data)

    flag = [1 for i in range(0, length)]
    for i in range(0, length):
        for j in range(i+1, length):
#            print(data[i], data[j])
            if gcd(data[i], data[j]) != 1:
                flag[j] = 0          # flag设置为0，标志为不可用
                flag[i] = 0
                break
        if flag[i] and (j == length - 1) and (gcd(data[i], data[j-1]) ==
1):
            res.append(data[i])
            print("res:", res)

    res = list(set(res))             # 去重
```

```
    print(res, len(res))
    return res

def main():
    url = './file/file_2018_input.txt'

    GenerateData(url)
    data = readFile(url)

    length = len(data)
    res = filter_list(data, length)

    writeFile(res, length)


main()
```

2018. 保研

1、编写一个函数，首先，产生一个100至200之间的随机整数rnd；然后，再产生rnd个100至500以内的随机整数，将这些整数保存在列表numberLst中。

2、编写一个函数，找出numberLst中所有包含数字2或数字6的整数，保存到列表num26Lst中。

3、将num26Lst中所有元素输出到屏幕，要求每行输出8个整数，每个整数占5列，右对齐。

4、编写一个函数，求出num26Lst中所有整数的因子，其中因子不包括1和整数本身，存放到另一个新的列表resultLst中。

5、编写一个函数，统计resultLst中每个因子出现的次数。

6、编写一个函数，将第5步的统计结果输出到屏幕，要求**每行一个统计结果，只输出出现次数最多的5个因子**。

7、编写一个函数，删除resultLst中每个因子的重复因子，每个因子只保留一份。

8、编写一个函数，将删除重复因子的resultLst列表输出到D盘文件result.txt中，要求每行输出8个整数，每个整数占5列，右对齐。

```
if __name__ == "__main__":
    # ----产生随机整数-------
    numberLst = productRndNum()
```

```
    # ----找出包含数字2或6的整数，其中digLst包含数字2和6-----
    num26Lst = getDigNumber(numberLst, digLst=[2， 6])
        printOut(num26Lst, 8)
    #-----找出所有整数的因子-----
    resultLst = getDivisorNum(num26Lst)
    #-----统计每个因子出现的次数-----
    resultStatic = staticResult(resultLst)
    printMax5Out(resultStatic)
    # ----删除resultLst中重复因子的多余份数，只保留一份-----
    delMultiDivisor(resultLst)
    print("===出现次数最多的数字===")
    printDivisorToFile("d:\\result.txt", resultLst)
```

代码：

```
# -*- coding: utf-8 -*-
import random
# 产生一个100-200之间的随机整数 rmd
# 再产生rmd个100至500以内的随机整数，保存到numberLst
def productRndNum():
    random.seed(100)
    rmd = random.randint(100, 200)  # [100,200]
    numberLst = [random.randint(100, 500) for i in range(rmd)]
    return numberLst

# ----找出包含数字2或6的整数，其中digLst包含数字2和6-----
def getDigNumber(numberLst, digLst):
    num26Lst = []
    for nb in numberLst:
        nstr = str(nb)
        for e in digLst:
            if str(e) in nstr:
                num26Lst.append(nb)
                break
    return num26Lst

# 每行输出8个整数，每个整数占5列，右对齐
def printOut(num26Lst, cnt):
    nlen = len(num26Lst)
    for i in range(nlen):
        print("{0:>5}".format(num26Lst[i]), end=' ')
        if (i + 1) % cnt == 0:
```

```
            print('')
    print('')

#-----找出所有整数的因子，因子不包括1和整数本身-----
def getDivisorNum(num26Lst):
    resultLst = []
    for numb in num26Lst:
        t = numb
        i = 2
        while i < t:
            if t % i == 0:
                resultLst.append(i)
            i = i + 1
    return resultLst

# 统计resultLst中每个因子出现的次数
def staticResult(resultLst):
    freq = {}
    for num in resultLst:
        freq[num] = freq.get(num, 0) + 1
    return freq

def printMax5Out(resultStatic):
    resultStatic = sorted(resultStatic.items(), key=lambda x:(x[1]),
reverse=True)
    for elem in resultStatic[:5]:
        print("{0} : {1}".format(elem[0], elem[1]))

def delMultiDivisor(resultLst):
    resultLst = sorted(set(resultLst), key=resultLst.index)
    print(resultLst)

# 将删除重复因子的resultLst列表输出到D盘文件result.txt中，
# 要求每行输出8个整数，每个整数占5列，右对齐
def printDivisorToFile(url, resultLst):
    with open(url, 'w', encoding='utf8') as f:
        rlen = len(resultLst)
        for i in range(rlen):
            f.write('{0:>5}'.format(resultLst[i]))
            if (i + 1) % 8 == 0:
                f.write('\n')
        f.write('\n')
```

```
if __name__ == "__main__":
    # ----产生随机整数-------
    numberLst = productRndNum()
    # ----找出包含数字2或6的整数，其中digLst包含数字2和6-----
    num26Lst = getDigNumber(numberLst, digLst=[2, 6])
    printOut(num26Lst, 8)
    #-----找出所有整数的因子-----
    resultLst = getDivisorNum(num26Lst)
    # -----统计每个因子出现的次数-----
    resultStatic = staticResult(resultLst)
    printMax5Out(resultStatic)
    # ----删除resultLst中重复因子的多余份数，只保留一份-----
    delMultiDivisor(resultLst)
    print("===出现次数最多的数字===")
    printDivisorToFile("./file/file_yan2018_result.txt", resultLst)
```

2019. 真题 因子拆分

请各位考生从考试信息发布网站下载数据文件data.txt。

已知：数据文件data.txt是一个文本文件，其中存放了 100个 不超过32768 的 **非负整数。**

请编写程序完成如下功能：

(1) 编写函数 read_file 从文件中 **读取数据**，将 **所有的整数** 按照 其 **在文件中出现的顺序** 依次存储到数组arr中；

(2) 编写函数print：**将数组arr显示在屏幕上**， **每行显示n个数**， **每个整数占6列**；

(3) 编写函数count： **统计数字0至9** 在数**组arr**所有整数中的**出现次数**，将**结果放入数组res**中（即res[0]存储数字0的出现次数，res[1]存储数字1的出现次数，其余以此类推）；

(4) 编写函数print_res：**将数组res显示在屏幕上，每行显示5个数**，可以复用步骤(2)中print函数；

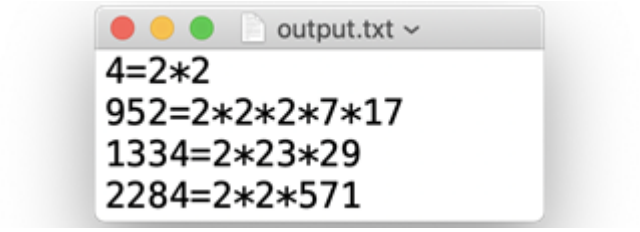
(5) 编写函数sort_array ：将数组arr中的整数 按照 **因子和 从小到大排序**，如果两个整数的因子和相等，则按照它们的自然大小排序（注意：计算一个整数的因子和时 **包括1和其本身**）；

(6) 编写函数filter_array： 对数组arr中的整数 进行 **筛选**，结果继续保存在arr中，**筛选规则**如下：

- 保留所有的偶数，同时保证这些偶数按照从小到大排序。
- 说明：完成筛选之后，数组arr中的元素可以分成两部分：

- 前半部分是有效内容，即**所有的偶数**
- 后半部分则是无效内容，参数size记录了数组arr中有效内容的长度（**注意**：筛选要求在原数组上进行）；

(7) 编写函数 **write_file** 对 数组arr中的有效内容（即所有偶数）进行**质因数分解**，并将 结果输出到屏幕和文本文件output.txt中。输出要求：**每一个整数的质因数分解结果占一行**，具体显示格式如下图所示：



代码：

```
# -*- coding: utf-8 -*-
import random
import re

def generateData(url):
    random.seed(1)
#    res = [random.randint(0, 10) for i in range(0, 10)]   # 测试
    res = [random.randint(0, 2768) for i in range(0, 100)]
    with open(url, 'w', encoding='utf8') as f:
        for elem in res:
            f.write(str(elem) + " ")
            if (res.index(elem) + 1) % 10 == 0:
                f.write('\n')

# 读文件
def read_file(url):
    with open(url, 'r+', encoding='utf8') as f:
        data = f.read().split()
        data = list(map(int, data))
    return data

# 打印数组
def print_arr(alist, n):
    length = len(alist)
    for i in range(0, length):
        if i == 0:
```

```
            pass
        elif i % n == 0:
            print("")
        print("{0:6}".format(alist[i]), end='')

    print('\n')

# 统计数字0至9  在数组arr所有整数中的出现次数
def count(alist):
    res = [0 for i in range(0, 10)]
    length = len(alist)

    for i in range(0, length):
        num = list(str(alist[i]))    # 例: 123--> ['1', '2', '3']
        n_len = len(num)
        for j in range(0, n_len):
            res[int(num[j])] += 1
#    print(res)
    return res

# 因子和
def factor(n):
    res = 0
    for i in range(1, n+1):
        if n % i == 0:
            res += i
    return res

# 按照 因子和  从小到大排序，如果两个整数的因子和相等，则按照它们的自然大小排序
def sort_array(alist):
    alist.sort(key=lambda x:(factor(x), x))
    return alist

def filter_array(alist):
    arr = list(filter(lambda x:x%2==0, alist))
    arr.sort()
    arr.append(len(arr))
    return arr

def filter_fac(n):
    i = 2
    res = []
```

```
    while i <= n:
        if n % i == 0:
           res.append(i)
           n /= i
        else:
           i = i + 1
#    print(res)
    return res

def write_file(arr):
    with open('./file/file_2019_out.txt', 'w', encoding='utf8') as f:
        for elem in arr:
            fac = filter_fac(elem)
            fac = list(map(str, fac))
            print("{0}={1}".format(elem, "*".join(fac)))
            f.write("{0}={1}\n".format(elem, "*".join(fac)))

def main():
    url = './file/file_2019_in.txt'
    generateData(url)
    # 第一问
    data = read_file(url)
    print_arr(data, 10)
    # 第二问
    res = count(data)           # 统计数字0至9  在数组arr所有整数中的出现次数
    print_arr(res, 5)
    # 第(5)问
    data = sort_array(data)
    print(data)
    # 第(6)问
    arr = filter_array(data)
    print(arr)
    # 第(7)问
    write_file(arr)

main()
```

输出：

```
   550  2331   258  1044   482  2029  1841  1934  2668  1554
略
    38   103    92    23    37
[8, 88, 104, 116, 118, 178, 258, 354, 384, 418, 442, 482, 550, 670, 690,
708, 896, 908,
略
8=2*2*2
88=2*2*2*11
略
```

2019. 保研 正则表达式拆分字符串

按以下要求编写程序

代码：

```
# -*- coding: utf-8 -*-

# 现有一个文本文件data.txt，其中记录了车辆进出校园时在自动收费系统ETC中记录下的ETC编号和时间信息。
# 每个ETC编号唯一地对应于一辆机动车。试用python语言编写程序，按要求从该数据文件中提取所需的信息。
#
# 说明：
# 1．一个ETC编号由5部分构成，本别是：两个大写字母、一个短横线、三位数字、一个短横线、三位数字。
#     如：“FG-102-934”、“BA-724-433”等都是合法的ETC编号。
# 2．时间信息的记录格式为：2016-01-08#07:21:31。
# 3．整条ETC记录的格式由5部分构成，分别是：ETC编号、字符“|”、入校时间、字符“|”，离校时间。
#     如：“BA-724-433|2016-01-08#07:21:31|2016-01-08#17:01:09”就是一条结构完整的ETC记录。
# 4．数据文件中每一行为一个ETC记录。
# 5．车辆进入校园后都是在当天离开校园的，即每条ETC记录中出入校园的日期是相同的。
# 6．ETC记录中出现的字符都是英文符号，无汉字和中文标点符号。
#
# 要求：
# 1．从文件中识别ETC记录，计算总的ETC记录的条数。提示：读取文件，并将ETC记录放入列表中。
# 2．计算ETC记录中共有多少辆不同的车。提示：通过正则表达式（或字符串分片）识别每条ETC记录中的ETC编号，
```

```
#     并将ETC编号放入集合中。
# 3. 找出进出校园次数最多的5辆车。提示：构建ETC编号和出现次数的字典。
# 4. 找出在校园中累计停留时间最长的5辆车。提示：构建ETC编号和累计停留时间的字典。
#     可通过正则表达式（或字符串分片）识别时间字符串及其中的时、分、秒信息。
#     为简化计算，可将时间转化成以零点开始计算的秒数。可以把计算时间差作为一个独立的函数。
#5.       将上述计算结果按“report.txt文件内容示例”（见最后一页）所示的要求输出到report.txt文件中。

import re

# 读文件，获取全部ETC记录，构成列表
def get_record(url):
    with open(url, 'r+', encoding='utf8') as f:
        data = f.read()
        data = data.split('\n')
    return data[:-1]

# 获取全部不同的ETC编号，构成集合
def get_v(vehicle_lst):
    veh_set = set()
    for elem in vehicle_lst:
        veh = elem.split('|')[0]
        veh_set.add(veh)

    # 共有多少辆不同的车
#    print(len(veh_set))
    return veh_set

# 构造车辆进出校园次数的字典
def count_v(vehicle_lst, vehicle_set):
    vehicles = {}
    for elem in vehicle_lst:
        veh = elem.split('|')[0]
        vehicles[veh] = vehicles.get(veh, 0) + 1      # 遇到一次编码，次数+1，默认为0
    vehicles = sorted(vehicles.items(), key=lambda x:(x[1]), reverse=True)[:10]

    # 进出校园次数最多的5辆车
    print(dict(vehicles[:5]))
    return vehicles[:5]
```

```
# 问题4:
# 时间 转化成 秒
def time2int(time : str):
    time = time.split(':')
    hour = int(time[0])*3600
    minu = int(time[1])*60
    sec = int(time[2])
    return hour + minu + sec

# 得到秒后，再 转化成 时间
def sec2time(seconds):
    m, s = divmod(seconds, 60)
    h, m = divmod(m, 60)
    return "{0}:{1:02d}:{2:02d}".format(h, m, s)

# 计算时间差
def calcTimeDiff(start, end):
    return int(time2int(end) - time2int(start))

# 构造车辆累计停留时间的字典
def count_t(vehicle_lst, vehicle_set):
    vehicles = {}
    for elem in vehicle_lst:
        veh = elem.split('|')[0]
        time = re.findall('[0-9]+:[0-9]+:[0-9]+', elem)
        diff = calcTimeDiff(time[0], time[1])
        vehicles[veh] = vehicles.get(veh, 0) + diff   # 累加停留的所有时间

    vehicles = sorted(vehicles.items(), key=lambda x:(x[1]),
reverse=True)
    # 如果要把 秒 转化成 时:分:秒
#    vehicles = [(elem[0], sec2time(elem[1])) for elem in vehicles]

    # 累计停留时间最长的5辆车
    print(vehicles[:5])
    return vehicles[:5]


# vehicle_lst: 总的ETC记录的条数
# fre_dict: 进出校园次数最多的5辆车
# inter_dict: 进出校园次数最多的5辆车
```

```
def write_to_file(vehicle_lst, fre_dict, inter_dict, url):
    with open(url, 'w+', encoding='utf8') as f:
        # 总的 ETC记录的 条数
        f.write("记录条数: " + str(len(vehicle_lst)) + '\n')
        # 共有多少辆不同的车
        f.write("车辆数: " + str(len(get_v(vehicle_lst))) + '\n')
        # 进出校园次数最多的5辆车
        f.write("进校次数最多的5辆车（单位：次）\n")
        for elem in fre_dict:
            f.write("{0}, {1}\n".format(str(elem[0]), str(elem[1])))
        # 在校园中累计停留时间最长的5辆车
        f.write("累计停留时间最长的5辆车（单位：秒）\n")
        for elem in inter_dict:
            f.write("{0}, {1}\n".format(str(elem[0]), str(elem[1])))


def main():
    # 读文件，获取全部ETC记录，构成列表
    vehicle_lst = get_record("./file/file_2019_input.txt")
    # 获取全部不同的ETC编号，构成集合
    vehicle_set = get_v(vehicle_lst)
    # 构造车辆进出校园次数的字典
    fre_dict = count_v(vehicle_lst, vehicle_set)
    # 构造车辆累计停留时间的字典
    inter_dict = count_t(vehicle_lst, vehicle_set)
    # 输出结果到文件中
    write_to_file(vehicle_lst, fre_dict, inter_dict,
"./file/file_2019_report.txt")
    return

main()  # 调用main函数
```

输出：

```
记录条数: 645
车辆数: 103
进校次数最多的5辆车（单位：次）
BR-189-680, 20
SX-733-715, 19
QC-621-143, 18
VI-487-543, 17
TR-445-782, 16
```

累计停留时间最长的5辆车（单位：秒）
QC-621-143，828417
BR-189-680，756757
UI-370-311，722708
SX-733-715，701937
TR-445-782，697439

2016. 期中考试 提取字符串 坐标问题

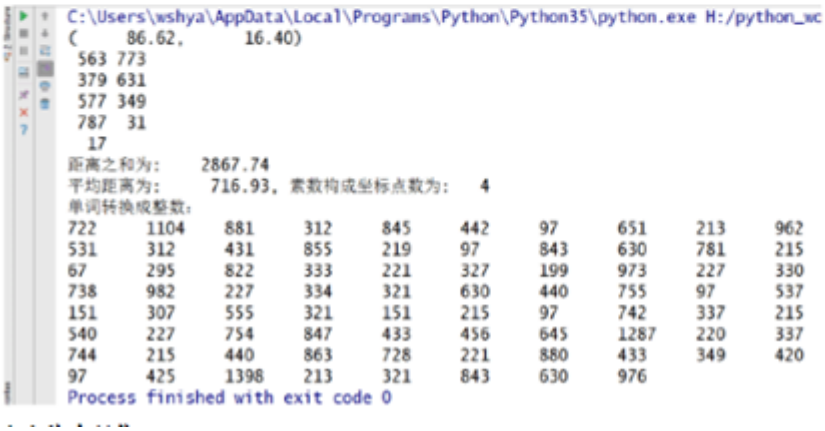
2. 提取第一步中 string 中包含数字 3 或数字 7 的所有素数，并将满足条件的素数显示在屏幕上，要求每个值占 10 列、右对齐，每行显示 2 个数。
   例如：整数 296 不是素数，563 是素数且包含数字 3
   提示：如果无法提取整数，可以自己设定几个素数(例如取 100 以内的素数)以完成后面的步骤，此步不得分，后面根据具体完成情况分步给分。
3. 将上述第二步中所生成的所有素数按顺序以两个数构成二维平面上点的坐标，如果最后存在单个素数，则丢弃。例如：素数：563、773、379、631，577，则(563,773)构成一个坐标点，(379,631)构成一个坐标点，577 因为是单个素数，则丢弃。
4. 产生两个[0,100]范围内的随机实数，以这个两个数构成二维平面上一个点 A 的坐标，并将 A 点坐标输出到屏幕上，要求以(x,y)的形式进行显示，其中 x 和 y 右对齐、占 10 列、保留 2 位小数。
5. 计算第三步找到的所有坐标点到点 A 之间的欧式距离之和。
6. 计算第三步找到的所有坐标点到点 A 之间的平均距离。
7. 举例：假设第一步产生的点 A 为(0,1)，第二步找到的坐标点为(563,773)、(379,631)，则距离之和为sumDistance = $\sqrt{653-0^2+(773-1)^2}+\sqrt{379-0^2+(631-1)^2}$;

平均距离为:avgDistance=sumDistance/2，(2 表示第三步找到的坐标点只有 2 个)

8. 将欧式距离之和以及平均距离显示在屏幕上，要求每个值输出占 10 列，保留 2 位小数。
9. (选做题，不计入考试分数，只供图灵班筛选)提取第一步 string 中的所有单词，其中连续的字母字符串称为一个单词，并将单词中所有字母的 ASCII 之和显示在屏幕上，要求每行显示 10 个整数，每个整数占 8 列，左对齐。

   例如：单词 Regular 对应的整数：R、e、g、u、l、a、r 的 ASCII 值之和。

一个简单示例结果(不是标准答案)：



```
C:\Users\wshya\AppData\Local\Programs\Python\Python35\python.exe H:/python_wc
(     86.62,      16.40)
 563 773
 379 631
 577 349
 787  31
  17
距离之和为:    2867.74
平均距离为:     716.93, 素数构成坐标点数为:   4
单词转换成整数:
722     1104    881     312     845     442     97      651     213     962
531     312     431     855     219     97      843     630     781     215
67      295     822     333     221     327     199     973     227     330
738     982     227     334     321     630     440     755     97      537
151     307     555     321     151     215     97      742     337     215
540     227     754     847     433     456     645     1287    220     337
744     215     440     863     728     221     880     433     349     420
97      425     1398    213     321     843     630     976
Process finished with exit code 0
```

评分标准

（编程题满分为80分）

大项	子项	评分项	应得分	实得分
正确性 70分	结果（70分）(程序无法运行则此项不得分)	产生两个随机实数	5	
		随机数构成的坐标点输出格式	5	
		提取所有整数	10	
		判断素数	10	
		判断素数包含数字 3 或 7	10	
		计算坐标点之间的距离	10	
		计算距离之和	5	
		计算平均距离	5	
		素数构成的坐标点输出格式	5	
		距离之和、平均距离输出格式	5	
		提取所有单词		
		将单词转换成整数		
		整数输出格式		

代码：

```
# -*- coding: utf-8 -*-
import math
```

```
import re
import random

# 判断素数
def is_prime(n):
    if n < 2:
        return False
    top = math.sqrt(n)
    i = 2
    while i <= top:
        if n % i == 0:
            return False
        i = i + 1
    return True

# 判断数中是否包含 3或7
def judge(n):
    li = list(n)
    return '3' in li or '7' in li

# 读文件
def readFile(url):
    with open(url, 'r+', encoding='utf8') as f:
        data = f.read()
        return data

# 生成坐标 A
def generateA():
    random.seed(100)
    a = random.uniform(0, 101)
    b = random.uniform(0, 101)
    return (a, b)

# 计算坐标点之间的距离
def distance(a, b):
    return math.sqrt(math.pow(a[0]-b[0], 2) + math.pow(a[1]-b[1], 2))

# 将单词转换成整数
def ASCII(word):
    w = list(word)
    res = 0
    for e in w:
```

```
        res += ord(e)
    return res

def main():
    data = readFile('./file/file_midterm.txt')
    print(data)

    # 提取所有整数
    num = re.findall('[0-9]+', data)
    # 判断素数 and 判断素数包含数字3或7
    primes = [int(elem) for elem in num if is_prime(int(elem)) and
judge(elem)]
    for elem in primes:
        # 每个值占10列、右对齐、每行显示2个数
        print("{0:>10}".format(elem), end=' ')
        if (primes.index(elem)) % 2:
            print('')

    print('')
    # 排序坐标
    primes.sort()
#    print(primes)

    plen = len(primes)
    plen = plen if (plen % 2 == 0) else plen - 1

    points = [(primes[i], primes[i+1]) for i in range(0, plen-1, 2)]
#    print(points)

    # 随机数构成的坐标点输出格式
    A = generateA()
    print("({0:>10.2f}, {1:10.2f})".format(A[0], A[1]))

    # 计算所有坐标点到A之间的欧式距离之和
    dis = []
    for i in range(0, plen//2):
        d = distance(points[i], A)
        dis.append(d)
    print("距离之和: {0:.2f}".format(sum(dis)))
    print("平均距离为:{0:10.2f}, 素数构成坐标点数为:
{1:10.2f}".format(sum(dis) / len(dis), plen//2))
```

```
    # 提取所有单词
    words = re.findall('[a-zA-Z]+', data)
    print(words)

    print("单词转化成整数:")
    wlen = len(words)
    ascii_words = [ASCII(elem) for elem in words]
    for i in range(0, wlen):
        print('{0:8}'.format(ascii_words[i]), end='')
        if (i+1) % 10 == 0:
            print('')

main()
```

输出：

```
       563        773
       379        631
略
(     14.71,      45.95)
距离之和: 2203.42
平均距离为:    550.85, 素数构成坐标点数为:      4.00
['Regular', 'expression', 'patterns', 'are', 'compiled', 'into', 'a',
'series', 'of', 'bytecodes', 'which', 'are', 'then', 'executed', 'by',
'a', 'matching', 'engine',
单词转化成整数:
     722    1104     881     312     845     442      97     651     213
962
     531     312     431     855     219      97     843     630     781
215
略
```

2017. 期中考试

现在有一文本文件data.txt，请将文本文件拷贝到D盘根目录，文本文件中包含多行字符串，每行字符串中包含若干用**空格、逗号、句号**分隔的单词，请编写程序完成如下功能：

- 编写一个函数，从data.txt文件中读取所有单词，并保存到单词列表**wordlst**中。
- 编写一个函数，找出wordlst中存在某个字母 **至少出现num次** 的单词，**字母不区分大小写**，将符合要求的单词保存到列表 **wordResultLst** 中，其中**num**由参数给出。

- 编写一个函数，删除wordResultLst中重复单词多余份数，只保留一份，**非重复单词保持不变**。将结果仍然保存在列表wordResultLst中。
- 编写一个函数，**输出wordResultLst中所有单词**，要求**每个单词占20列**，每行输出count个单词，其中count由参数给出。
- 编写一个函数，**将wordResultLst中的每一个单词转换成一个整数**，保存到列表numLst中。转换规则：整数为单词的所有字母的ASCII值的累加和，例如：sum对应的整数就是s、u、m三个字母的ASCII值之和。
- 编写一个函数，对numLst中的**所有整数按整数的数字累加和**进行降序排序，例如整数:923,456,134对应的整数数字累加和为14，15，8，则排序结果456，923，134。
- 编写一个函数，**输出排序后的numLst**，**要求每个整数占8列，每行输出count个整数**，其中count由参数给出。
- 编写一个函数，**统计numLst中每个数字出现的次数**，将统计结果保存到字典resultDic中。
- 编写一个函数，将统计结果resultDic中出现次数最多的数字及其出现次数输出到D盘根目录的文本文件result.txt中
    - 输出格式：**数字(占2列,左对齐) ：出现次数(占3列，右对齐)**

测试程序如下(不允许修改测试程序，修改一处扣2分，直到扣满10分)：

```
if __name__ == "__main__":
    # ----从data.txt文件中读取所有单词-------
    wordlst = readWordsFromFile("d:\\data.txt")
    print("文件中单词个数:", len(wordlst))  # 输出单词个数
```

```
spyderWorkspace/testOfLesson')
文件中单词个数: 65
至少含有重复2次的字母的单词:  13
===删除重复单词的多余单词后的结果===
That            statements      program         packaged
procedure       function        parameters      that
serve           difFer          Operation
===整数降序排序的结果===
  969   549   945 1096   592
  816   870 1076   760   433
  401
===出现次数最多的数字===
9 :  6
```

```
# -*- coding: utf-8 -*-
import re

def readWordsFromFile(url):
    with open(url, 'r+', encoding='utf8') as f:
        data = f.read()
        words = re.findall('[a-zA-Z]+', data)
    return words

def findMultiAlphaWords(wordlst, cnt):
    wordResultLst = []
    for word in wordlst:
        w = list(word.lower())
        for e in w:
            if w.count(e) >= cnt:
                wordResultLst.append(word)
                break
    print(wordResultLst)
    return wordResultLst

def delMultiData(wordResultLst):
    tmp = wordResultLst[::]
    for elem in tmp:
        if wordResultLst.count(elem) > 1:
            wordResultLst.remove(elem)

def printWordLst(wordResultLst, cnt):
    wlen = len(wordResultLst)
    for i in range(0, wlen):
        print('{0:20}'.format(wordResultLst[i]), end='')
        if (i + 1) % cnt == 0:
            print('')
    print('')

def ASCII(word):
    ans = 0
    for w in word:
        ans += ord(w)
    return ans

def getNumberOfWords(wordResultLst):
```

```
    numlst = []
    for word in wordResultLst:
        numlst.append(ASCII(word))
    return numlst

def numSum(x):
    li = str(x)
    res = 0
    for e in li:
        res += int(e)
    return res

def sortByDigitalSum(numlst):
    numlst.sort(key=lambda x:numSum(x), reverse=True)
    return numlst

def printNumLst(numlst, cnt):
    wlen = len(numlst)
    for i in range(0, wlen):
        print('{0:8}'.format(numlst[i]), end='')
        if (i + 1) % cnt == 0:
            print('')
    print('')

# ----统计数字出现的次数----------
def staticDigitalTimes(numlst):
    resultDic = {}
    for e in numlst:
        num = str(e)
        for w in num:
            resultDic[w] = resultDic.get(w, 0) + 1
    print(resultDic)
    return resultDic

def printDicToFile(url, resultDic):
    resultDic = sorted(resultDic.items(), key=lambda x:x[1],
reverse=True)
    with open(url, 'w', encoding='utf8') as f:
        print('{0:<2}:{1:>3}'.format(resultDic[0][0], resultDic[0][1]))
        f.write('{0:<2}:{1:>3}'.format(resultDic[0][0], resultDic[0][1]))

if __name__ == "__main__":
```

```
    # ----从data.txt文件中读取所有单词-------
    wordlst = readWordsFromFile("./file/file_midterm_2017.txt")
    print("文件中单词个数:", len(wordlst))  # 输出单词个数
    # ----找出单词中，存在某个字母重复num次的单词-----
    wordResultLst = findMultiAlphaWords(wordlst, 2)
    print("至少含有重复2次的字母的单词：", len(wordResultLst))
#
    # ----删除wordResultLst中重复单词的多余份数，只保留一份-----
    delMultiData(wordResultLst)
    print("===删除重复单词的多余单词后的结果===")
    printWordLst(wordResultLst, 4)  # 输出所有单词，每行输出4个单词

    # ----将wordResultLst中的所有单词转换为整数------
    numlst = getNumberOfWords(wordResultLst)

    # ----对numlst中的所有整数进行根据数字累加和进行降序排序----
    sortByDigitalSum(numlst)
    print("===整数降序排序的结果===")
    printNumLst(numlst, 5)    #输出整数列表，每行输出5个整数

    # ----统计数字出现的次数----------
    resultDic = staticDigitalTimes(numlst)

    print("===出现次数最多的数字===")
    printDicToFile("./file/file_midterm_2017.out", resultDic)
```

2018. 期中考试

1.

3. 一个正整数的头部和尾部分别是其第一位数字和最后一位数字。比如123的头部是1，尾部是3。5的头部和尾部都是5。给定一个正整数列表，将其中每个元素用它的头部和尾部进行替换，从而得到一个包含若干数字的列表T，将T中的质数保持不变并看成分隔符，可以把T分割成若干个子序列，对于每个子序列，将其中的数字进行合并得到一个新的数，返回合并之后的列表。比如，对于列表[1, 234, 5, 6, 70, 890]，替换之后的列表是[1, 1, 2, 4, 5, 5, 6, 6, 7, 0, 8, 0]，其中包含的子序列有<1,1>, <4>, <6,6>和<0,8,0>，合并之后的列表是[11, 2, 4, 5, 5, 66, 7, 80]。

相关说明	
输入条件	仅包含正整数的列表
输出要求	列表
其它要求	将代码写入函数 func3

测试用例：

输入	返回
[1, 234, 5, 6, 7, 890]	[11, 2, 4, 5, 5, 66, 7, 7, 80]
[12, 34, 56, 78, 90]	[1, 2, 3, 4, 5, 6, 7, 890]
[123]	[1, 3]

代码：

```
# -*- coding: utf-8 -*-
import math
from functools import reduce
def is_prime(num):
    if num < 2:
        return False
    top = int(math.sqrt(num))
    i = 2
    while i <= top:
        if num % i == 0:
            return False
```

```
        i = i + 1
    return True

def solve():
    li = eval(input())    # [1, 234, 5, 6, 70, 890], [12,34,56,78,90]
    rli = []
    for elem in li:
        estr = str(elem)
        rli.append(int(estr[0]))
        rli.append(int(estr[-1]))

    rlen = len(rli)
    flag = [0] * rlen
    ans = []
    for i in range(rlen):
        ps = []              # 是prime
        notps = []           # 不是prime
        if flag[i] == 0 and is_prime(rli[i]):
            for j in range(i, rlen):
                if flag[j] == 0 and is_prime(rli[j]):
                    ps.append(rli[j])
                    flag[j] = 1
                else:
                    break
            if ps:
                ans.append(ps)
        elif flag[i] == 0:
            for j in range(i, rlen):
                if flag[j] == 0 and not is_prime(rli[j]):
                    notps.append(rli[j])
                    flag[j] = 1
                else:
                    break
            if notps:
                ans.append(notps)

    res = []
    for elem in ans:
        comb = reduce(lambda x,y:x*10+y, elem)
        res.append(comb)
    print(res)
```

```
solve()
```

代码：

```
# -*- coding: utf-8 -*-
import collections
import itertools

def func4(mat):
    m=len(mat)
    mm=[[0 for i in range(m)] for j in range(2*m-1)]
    for i in range(m):
        for j in range(m):
            if i+j<m:
                mm[i+j][j]=mat[i][j]
            else:
                mm[i+j][m-i-1]=mat[i][j]
    return mm

def solve():
    A = eval(input()) # [[1,2,3], [4,5,6], [7,8,9]]
    m = len(A)
    B = [[0]*m for i in range(2*m-1)]
    row = 2*m - 1

    dic = collections.defaultdict(list)
    for i,j in itertools.product(range(m), range(m)):
        dic[i - j].append(A[m-i-1][j])

    dic = sorted(dic.items(), key=lambda x:x[0], reverse=True)
    for i in range(row):
        rlen = len(dic[i][1])
        for j in range(rlen):
            B[i][j] = dic[i][1][j]
    print(B)

solve()
print(func4([[1,2,3], [4,5,6], [7,8,9]]))
```

词，那么就认为小明掌握了这个单词。注意：每次拼写时，chars 中的每个字母都只能用一次。返回 words 中小明所掌握的单词个数。

相关说明	
输入条件	words 中的每一个字符串长度一定大于等于 1 chars 的长度一定大于等于 1
输出要求	返回一个整数
其它要求	将代码写入函数 func7

建议测试用例：

输入	返回
words = ["cat","bt","hat","tree"], chars = "atach"	2 解释： 可以形成字符串 "cat" 和 "hat"
words = ["hello","world","soochow"], chars = "welldonehoneyr"	2 解释： 可以形成字符串 "hello" 和 "world"

```
# -*- coding: utf-8 -*-

# 字母表中字幕出现的次数 >= 单词的字幕出现次数
def contains(charCount, wordCount):
    print(charCount, wordCount)
    for key in wordCount.keys():
        print(charCount.get(key, 0) , wordCount.get(key))
        if charCount.get(key, 0) < wordCount.get(key):
            return False
    return True

def solve(words, chars):
    from collections import Counter
    charCount = dict(Counter(chars))
    res = 0
    wlen = len(words)
    for i in range(0, wlen):
        wordCount = dict(Counter(words[i]))
        if contains(charCount, wordCount):
```

```
            res += 1
    return res

print(solve(['cat', 'bt', 'hat', 'tree'], 'atach'))
print(solve(['hello', 'world', 'soochow'], 'welldonehoneyr'))
```

期末考试

题目1 数组每个数出现次数

- 给你一个整数列表lst，请你帮忙统计数组中每个数的出现次数。
- 如果每个数的出现次数都是独一无二的，就返回 true；否则返回 false。
- 输入：lst = [1,2,2,1,1,3] 输出：true 解释：在该数组中，1 出现了 3 次，2 出现了 2 次，3 只出现了 1 次。没有两个数的出现次数相同。

代码：

```
# -*- coding: utf-8 -*-
def func2(lst : list):
    flag = {}
    for e in lst:
        flag[e] = flag.get(e, 0) + 1

    res = []
    for e in flag.values():
       res.append(e)

    rlen = len(res)
    for i in range(0, rlen):
        for j in range(i+1, rlen):
            if res[i] == res[j]:
                return False
    return True

def main():
    print(func2([1,2,2,1,1,3]))
    print(func2([1,2,2,1,1,3,3]))
    print(func2([1,2,2,1,1,3, 4,4,4]))
    print(func2([1,2,2,1,1,3, 4,4,4,4,4]))

main()
```

输出：

```
True
False
False
True
```

题目2 奇数在前 偶数在后

- 给定一个非负整数列表 lst，返回 lst 的排序结果，排序要求 首先是 **奇数在前，偶数在后**
- 然后，按照数字从大到小排序
- 示例： 输入：lst = [1,2,3,4,5,6] 输出： [1,3,5,2,4,6]

```
# -*- coding: utf-8 -*-
def fun2(lst):
    length = len(lst)
    i = 0
    j = length - 1
    while i < j:
        while i < j and lst[j] % 2 == 0:    # 后面是偶数
            j = j - 1
        if i < j:
            t = lst[j]
        while i < j and lst[i] % 2 == 1:    # 前面是奇数
            i = i + 1
        if i < j:
            lst[j] = lst[i]
            lst[i] = t
    print(lst)

def main():
    fun2([1,2,3,4,5,6])
    fun2([1, 2, 3, 6, 7, 10, 11])
    fun2([2, 3, 4, 5, 1, 3, 5, 7, 9])

main()
```

输出：

```
[1, 5, 3, 4, 2, 6]
[1, 11, 3, 7, 6, 10, 2]
[9, 3, 7, 5, 1, 3, 5, 4, 2]
```

题目3 查找添加的字母

- 给定两个字符串 s 和 t，它们 **只包含小写字母**。
- **字符串 t 由字符串 s 随机重排**，然后在随机位置添加一个字母。请找出在 t 中被添加的字母。
- 示例:

```
输入：
s = "abcd"
t = "abcde"
输出：
e
解释：
'e' 是那个被添加的字母。
```

代码：

```
# -*- coding: utf-8 -*-
# 给定两个字符串 s 和 t，它们只包含小写字母。字符串 t 由字符串 s 随机重排，
# 然后在随机位置添加一个字母。请找出在 t 中被添加的字母。
def func4(str1, str2):
    s = sorted(str1)
    t = sorted(str2)
    slen = len(s)
    tlen = len(t)

    for i in range(0, slen):
        if s[i] != t[i]:
            return t[i]
    return t[tlen - 1]

def main():
    print(func4('abcd', 'abcde'))
    print(func4('abcd', 'aebcd'))
    print(func4('abcd', 'abecd'))
    print(func4('abbcd', 'abbecb'))
```

```
main()
```

苏州大学python课程组习题

1. 因子和

编写一个函数，计算一个整数的所有因子之和，其中因子不包括整数本身，并编写测试程序，在测试程序中输入整数和输出整数的所有因子之和。例如：输入 8，调用该函数之后，得到结果为 7。

代码：

```
# -*- coding: utf-8 -*-
def factor(num):
    i = 1
    res = 0
    while i < num:
        if num % i == 0:
            res += i
        i = i + 1
    return res


def main():
    while True:
        n = int(input("输入n:"))
        print(factor(n))

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

2. 反素数

(反素数)反素数：指一个素数将其逆向拼写后也是一个素数的非回文数。 例如：17 和 71 都是素数且都不是回文数，所以 **17 和 71 都是反素数**。请编写一个函数判断一个数是否是反素数？并编写测试程序找出前 30 个反 素数输出到屏幕上，**要求每行输出 8 个数，每个数占 5 列，右对齐**。

代码：

```
# -*- coding: utf-8 -*-
# (反素数)反素数指一个素数将其逆向拼写后也是一个素数的非回文数。
# 例如：17 和 71 都是素数且都不是回文数，所以 17 和 71 都是反素数。
# 请编写一个函数判断一个数是否是反素数？并编写测试程序找出前 30 个反
# 素数输出到屏幕上，要求每行输出 8 个数，每个数占 5 列，右对齐。

import math
# 素数
def is_prime(num):
    if num < 2:
        return False
    top = math.sqrt(num)
    i = 2
    while i <= top:
        if num % i == 0:
            return False
        i = i + 1
    return True

def Reverse(num):
    return int(str(num)[::-1])

def print_res(res, length):
    for i in range(0, length):
        print("{0:>5}".format(res[i]), end='')
        if (i+1)%8 == 0:
            print('')
    print('')

def main():
    cnt = 0
    cur = 2
    res = []
    while cnt < 30:
        if is_prime(cur) and is_prime(Reverse(cur)):
            res.append(cur)
            cnt = cnt + 1
        cur = cur + 1

    print_res(res, len(res))
```

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

3. 梅森素数

【函数】(梅森素数) 如果一个素数 可以 写成 2^p-1 形式，其中 p 是一个正整数，那么该数就称作梅森素数。请编写一个函数：

- 判断一个素数是否是梅森素数，如果是，则返回 p 的值，否则返回-1。
- 并编写测试程序找出 1000 以内的所有梅森素数输出到屏幕上，要求输出格式如下：
- P(占 3 列右对齐) 2^p-1 (占 4 列右对齐) # 此行不需要输出

```
2  3       2^2-1
3  7       2^3-1
5  31      2^5-1
```

代码：

```
# -*- coding: utf-8 -*-
import math

def is_prime(num):
    if num < 2:
        return False
    top = math.sqrt(num)
    i = 2
    while i <= top:
        if num % i == 0:
            return False
        i = i + 1
    return True

def is_MalPrime(num):
    return is_prime(num) and is_prime(2**num-1)

def filter_prime():
    cur = 2
    res = []
    # 注意是 2^cur <= 1000
    while math.pow(2,cur) <= 1000:
        if is_MalPrime(cur):
```

```
            res.append(cur)
        cur = cur + 1
    return res

def print_res(res, length):
    for i in range(0, length):
        print("{0:>3} {1:>4}".format(res[i], 2**res[i]-1))

def main():
    res = filter_prime()
    print_res(res, len(res))

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

输出：

```
  2    3
  3    7
  5   31
  7  127
```

4. 加密函数

编写一个**加密函数**：

- 实现对一个**给定字符串** 中的**字母**，**转变** 为其后 n个字符，如果遇到 超过字母边界，则 从最小字母继续计数
- **连续 的 数字字符**，作为一个整数 扩大 n 倍之后，替换到对应位置，其中 n 默认为 5。
- 再编写一个**解密函数**： 实现对上述加密字符串进行解密。
- 编写测试程序，在测试程序中输入字符串，并**输出加密** 和 **解密后** 的字符串。

```
例如：
字符串 str1: avbV125av1, n 默认为 5
则新的字符串 str2: fagA625fa5
```

代码：

```
# -*- coding: utf-8 -*-
import re
# 生成字母
```

```
def generateAlpha():
    alpha_up = []
    alpha_low = []
    for i in range(26):
        alpha_up.append(chr(ord('A') + i))
    for i in range(26):
        alpha_low.append(chr(ord('a') + i))

    return [alpha_low, alpha_up]

# 加密
def encrapy(str1, n, alpha):
    alpha_low = alpha[0]      # 小写字母表
    alpha_up = alpha[1]       # 大写字母表
    length = len(str1)
    res = ""

    i = 0
    while i < length:
        num = ""
        if str1[i].isupper():
            res += alpha_up[(alpha_up.index(str1[i]) + n)%26]

        elif str1[i].islower():
            res += alpha_low[(alpha_low.index(str1[i]) + n)%26]

        elif str1[i].isnumeric():
            num += str1[i]
            i = i + 1
            while i < length and str1[i].isnumeric():
                num += str1[i]
                i = i + 1
            # 字符后退一个
            i = i - 1
            # 数字*n
            res += str(int(num)*n)
        i = i + 1

    return res

# 解密
def decode(str1, n, alpha):
```

```
    alpha_low = alpha[0]      # 小写字母表
    alpha_up = alpha[1]       # 大写字母表
    length = len(str1)
    res = ""

    i = 0
    while i < length:
        num = ""
        if str1[i].isupper():
            res += alpha_up[(alpha_up.index(str1[i]) - n)%26]

        elif str1[i].islower():
            res += alpha_low[(alpha_low.index(str1[i]) - n)%26]

        elif str1[i].isnumeric():
            num += str1[i]
            i = i + 1
            while i < length and str1[i].isnumeric():
                num += str1[i]
                i = i + 1
            # 字符后退一个
            i = i - 1
            res += str(int(num)//n)
        i = i + 1

    return res

def main():
    alpha = generateAlpha()

    str1 = 'avbV125av1'
    n = 5
    print("字符串:{0}, n默认为:{1}".format(str1, n))

    str2 = encrapy(str1, n, alpha)
    print("加密后:{0}".format(str2))

    str1 = decode(str2, n, alpha)
    print("解密后:{0}".format(str1))

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

方式2：

```
# -*- coding: utf-8 -*-
import re

# 加密
def encrapy(str1, n):
    words = re.findall('\d+|[a-zA-Z]+', str1)
    res = ''

    for word in words:
        if word.isnumeric():
            res += str(int(word)*n)
        elif word.isalpha():
            for c in word:
                k = ord(c) + n
                if ord('Z') < k < ord('a') or k > ord('z'):
                    res += chr(k - 26)
                else:
                    res += chr(k)

    return res

# 解密
def decode(str1, n):
    words = re.findall('\d+|[a-zA-Z]+', str1)
    res = ''

    for word in words:
        if word.isdigit():
            res += str(int(word)//n)
        elif word.isalpha():
            for c in word:
                k = ord(c) - n
                if k < ord('A') or ord('Z') < k < ord('a'):
                    res += chr(k + 26)
                else:
                    res += chr(k)
    return res

def main():
```

```
    str1 = 'avbV125av1'
    n = 5
    print("字符串:{0}, n默认为:{1}".format(str1, n))

    str2 = encrapy(str1, n)
    print("加密后:{0}".format(str2))

    str1 = decode(str2, n)
    print("解密后:{0}".format(str1))

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

输出：

```
字符串:avbV125av1, n默认为:5
加密后:fagA625fa5
解密后:avbV125av1
```

5. 正则表达式

请利用正则表达式写一个简单的拼写检查程序。实现以下功能： a) 两个或两个以上的空格出现时将其压缩为一个。 b) 在标点符号后加上一个空格，如果这个标点符合之后还有字母。 例：给定字符串："This□□is□□very□funny□and□□□cool.Indeed!" 输出："This□is□very□funny□and□cool.□Indeed!" 其中“□”代表一个空格。

代码：

```
# -*- coding: utf-8 -*-
import re
def readFile(url):
    with open(url, 'r+', encoding='utf8') as f:
        data = f.read()
        return data

def lessSpace(s):
    words = re.findall('[a-zA-Z]+|[.?!]', s)
    print(words)
    res = ''
    for word in words:
```

```
        if word.isalpha():
            res += word + ' '
        else:
            res = res[:-1]
            res += word + ' '

    if not res[-2].isalpha():
        return res[:-1]
    else:
        return res

def main():
    data = readFile('./file/file10_space.txt')
    words = lessSpace(data)
    print(words)

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

```
['This', 'is', 'very', 'funny', 'and', 'cool', '.', 'Indeed', '!', 'I',
'love', 'you', '.']
This is very funny and cool. Indeed! I love you.
```

6. 正则表达式 findall

请利用正则表达式写一个 Python 程序以尝试解析 XML/HTML 标签。现有 如下一段内容：

```
<composer>Wolfgang Amadeus Mozart</composer>
<author>Samuel Beckett</author>
<city>London</city>
希望自动格式化重写为：
composer: Wolfgang Amadeus Mozart
author: Samuel Beckett
city: London
```

代码：

```
# -*- coding: utf-8 -*-
import re

def readFile(url):
```

```
    with open(url, 'r', encoding='utf8') as f:
        data = f.read()
    return data

def parseXML(s):
    res = re.findall('<(.*)>(.*)</.*>', s)
    for word in res:
        print("{0}: {1}".format(word[0], word[1]))

def main():
    data = readFile('./file/file11_parse.txt')
    print(data)
    parseXML(data)

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

输出：

```
<composer>Wolfgang Amadeus Mozart</composer>
<author>Samuel Beckett</author>
<city>London</city>

composer: Wolfgang Amadeus Mozart
author: Samuel Beckett
city: London
```

59. 文件：有一个 100G 的文件 largefile.txt（这个文件目前没有100G，只是做模拟）。实现一个程序，首先输出 largefile.txt 的行数，然后无限循环，每次要求用户键盘输入一个行号，然后立刻输出对应行的文本。由于文件很大，不允许将文件内容全部放到内存中； 同时也不允许从头扫描文件，得到对应行的文本，因为这样速度太慢。（提示：用二进制模式打开文件，使用 tell, seek 等方法）

```
# -*- coding: utf-8 -*-
def readFile(url):
    with open(url, 'rb') as f:
        print("行号", len(f.readlines()))
        while True:
            line = ''
            f.seek(0)
            cur = 0
```

```
            n = int(input("input: "))
            while cur < n - 1:
                f.readline()
                cur += 1
            line = f.readline()
            if not line:
                break
            else:
                print(line)

def readFile1(url):
    count = 1
    with open(url, 'rb') as f:
        while True:
            buff = f.read(8192*1025)
            buff = buff.decode('utf8')
            print(buff)
            count += buff.count('\n')
            if not buff:
                break
    print(count)

def main():
#    readFile('./file/file15_largefile.txt')
    readFile1('./file/file15_largefile.txt')

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

MOOC测验

5.1 小朋友排队

```
"""
小明将学生排队   学号小到大排一排   然后进行多次调整
一次调整可能让一名同学出队 向前或先后移动一段距离再入队

例： 学生人数8人
0)初始 1,2,3,4,5,6,7,8
1)第一次调整 3号向后移动2    1,2,4,5,3,6,7,8
"""
```

```
class Solution():
    def MovingResult(self, m, lst):
        """
        m: 学生的数量
        lst: 每一个元素是一个元组: 元组的第一个元素是学号 第二个是移动的数量 负数表示向前
        返回一个列表 按照顺序存储了当前位置上学生的学号[1,2,3,4,5,6,7,8]
        """
        nums = [i for i in range(1, m+1)]
        nlen = len(lst)
        for i in range(nlen):
            op = lst[i]
            t = op[0]
            idx = nums.index(op[0])
            if op[1] > 0:
                for j in range(idx+1, idx + op[1]+1):
                    # print('nums[j] -> nums[j-1]', nums[j], nums[j-1])
                    nums[j-1] = nums[j]
            else:
                for j in range(idx-1, idx+op[1]-2, -1):
                    # print('nums[j] -> nums[j+1]', nums[j], nums[j+1])
                    nums[j+1] = nums[j]

            nums[idx + op[1]] = t
            # print('nums:', nums)
        return nums

s = Solution()
print(s.MovingResult(m = 8, lst = [(3, 2), (8, -3), (3, -2)]))
```

5.2 小朋友出队

- n个小朋友围一圈，小朋友从1~n编号。顺时针方向123...n12
- 游戏开始从1号开始顺时针报数，每个小朋友报上个小朋友数+1
- 若一个小朋友的数为k的倍数或其个位数为k，则该小朋友出去，不再参加以后的报数
- 当游戏中只剩下一个小朋友的时候该小朋友获胜

```
class Solution:
    def Circleplay(self, n, k):
        player = [i for i in range(n)]   # 0 ~ n-1
        cur = 0     # 当前报号
```

```
        num = 0
        while len(player) > 1:
            num += 1
            if num % k == 0 or num % 10 == k:
                del player[cur]
                cur = cur % len(player)    # 小朋友走的时候，不需要移动 cur
            else:
                cur = (cur + 1) % len(player)

        return player[0] + 1

s = Solution()
print(s.Circleplay(5, 2))
print(s.Circleplay(7, 3))
```