



# 实验6 控制结构2

ANDYWWW

LAST MODIFIED: 17 APR 2019

1. 请输出1988年与1989年，这两年里的日期所组成的素数。

**思路：**

- (1) 生成所有1988年、1989年日期组成的列表
- (2) 依次循环，判断列表内的数字是否为素数

**输出部分：**可以在（2）循环内判断后直接输出；也可以把所有的素数放到一个列表里，在所有日期判断完之后再对列表循环输出

1.请输出1988年与1989年，这两年里的日期所组成的素数。

答案：

```
import math
lst_rn = [31,29,31,30,31,30,31,31,30,31,30,31]  闰年月份
lst_pn = [31,28,31,30,31,30,31,31,30,31,30,31]  平年月份
for month in range(1,13):
    for day in range(1,lst_rn[month - 1] + 1):
        date = int("{}{:0>2d}{:0>2d}".format(1988,month,day))
        flag = 1
        for i in range(2,int(math.sqrt(date)) + 1):
            if date % i == 0:
                flag = 0
                break
        if flag:
            print(date)
```

也可以使用  
for...else...  
语句

这里是利用格式化模板转换日期对应的数字，也可以直接利用数学方法（见后部分代码）

直接用date \*\* 0.5也可以

1.请输出1988年与1989年，这两年里的日期所组成的素数。

答案： (接上)

```
for month in range(1,13):  
    for day in range(1,lst_pn[month - 1] + 1):  
        date = 1989 * 10000 + month * 100 + day  
        flag = 1  
        for i in range(2,int(math.sqrt(date)) + 1):  
            if date % i == 0:  
                flag = 0  
                break  
        if flag:  
            print(date)
```

#### 如何使用flag?

如果碰上此类的问题，考虑使用flag：某变量在满足某个性质时，取a；如果始终不满足则取b，这时候**初始化flag为b**。

这个时候flag可以作为一个标记，用来标记这个变量是否满足这个性质。

比如在素数判断中，初始化flag为1，如果这个数可以被一个数整除，那么flag转为0。如果在整个循环中flag始终都是1，则代表这个数是素数，如果flag只要变成了0，则这个数就是合数。因此flag == 1代表这个数是素数，flag == 0代表这个数是合数

# 1. 请输出1988年与1989年，这两年里的日期所组成的素数。

Ex. 这里有一系列学生成绩存在列表lst中，列表中每个元素也是一个列表，储存着这个学生的所有必修课成绩。现规定凡是有一门课低于60的学生需要做退学处理，请输出所有被退学学生在lst中的索引，如果没有被退学学生，则输出'not found'。

```
failed_lst = []
index = 0
for stu_score in lst:
    flag = 0          注意初始化的位置!
    for score in stu_score:
        if score < 60:
            flag = 1
            break      这里可以不break
    if flag:
        failed_lst.append(index)
    index += 1
```

```
if failed_lst:
    print(failed_lst)
else:
    print('not found')
```

if后的变量、表达式都会转化为Bool型再进行判断  
常见转为Bool型为False的变量有：空字符串（区别空格）、空列表（如本例）、0；与之对应的为True的有：  
非空字符串、非空列表、非零数

因此如果是not []、not ""，结果都为True

2. 输出所有的"水仙花数"。所谓"水仙花数"是指这样的一个三位数：其各位数字的立方和等于该数本身。

---

思路：

(1) 循环100-999，对于每个数进行分析。

(2) 将数字的每位上的数字提取出来，计算该数字

是否是水仙花数

Q：如何提取数字的某（几）位数字？

M1：将数字转成字符串，利用切片提取（仅限Python）

M2：利用数学方法

## 2. 输出所有的"水仙花数"。所谓"水仙花数"是指这样的一个三位数：其各位数字的立方和等于该数本身。

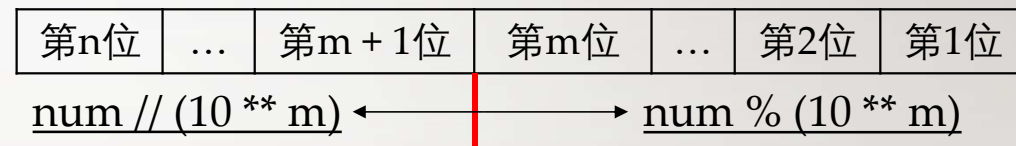
Ex. 从键盘键入一个数字（不知道具体位数），请将数字的每一位存到一个列表里，并输出这个数字各个位数的和。

```
num = input("Please input the number:")
num_lst = [int(i) for i in num]
print(sum(num_lst))
```

字符串切片

```
num = int(input("Please input the number:"))
num_lst = []
while num != 0:
    num_lst.append(num % 10)
    num = num // 10
print(sum(num_lst))
```

数学方法



num % 10用来提取num的个位数字。  
num // 10用来提取num的除了个位数字之外的数字  
直到num为0的时候，循环结束



2. 输出所有的"水仙花数"。所谓"水仙花数"是指这样的三位数：其各位数字的立方和等于该数本身。

Ex. 已知一列表内储存着一系列身份证号（整型），请将所有身份证号中的出生日期提取出来。

```
for ID in ID_lst:  
    date = str(ID)[6:-4]  
    print(date)
```

字符串切片

```
for ID in ID_lst:  
    date = (ID // (10 ** 4)) % (10 ** 8)  
    print(date)
```

数学方法





2. 输出所有的"水仙花数"。所谓"水仙花数"是指这样的一个三位数：其各位数字的立方和等于该数本身。

答案：

```
for num in range(100,1000):  
    str_num = str(num)  
    test_sum = int(str_num[0])**3 + int(str_num[1])**3 + int(str_num[2])**3  
    if test_sum == num:  
        print(num)
```

字符串切片

```
for num in range(100,1000):  
    test = 0; nnum = num  
    while num != 0:  
        test += (num % 10) ** 3  
        num = num // 10  
    if test == nnum:  
        print(nnum)
```

循环中num会改变，nnum用来备份数据

数学方法

3. 韩信点兵，让士兵每三人、五人、七人站一排，只看最后一排的人数就可以知道总人数（总人数不少于10，不大于100）

---

**思路：**

(1) 人数从10循环到100

(2) 判断人数除以3、5、7的余数是否都与输入相等

输出部分：可以在判断成功后输出结果并跳出循环（这时为了判断是否有解需要引入flag），也可以将所有满足条件的人数存在一个列表内，在循环外输出列表最小值。

3. 韩信点兵，让士兵每三人、五人、七人站一排，只看最后一排的人数就可以知道总人数（总人数不少于10，不大于100）

答案：

```
x = input()
while x != "":
    a = int(x.split()[0])
    b = int(x.split()[1])
    c = int(x.split()[2])
    flag = 0
    for people in range(10,101):
        if (people % 3 == a) and (people % 5 == b) and (people % 7 == c):
            flag = 1
            print(people)
            break
    if flag == 0:
        print('No answer')
    x = input()
```

result\_lst = []

result\_lst.append(people)

if result\_lst:  
 print(min(result\_lst))  
else:  
 print('No answer')

浅蓝底的代码  
是另一种方法

3. 韩信点兵，让士兵每三人、五人、七人站一排，只看最后一排的人数就可以知道总人数（总人数不少于10，不大于100）

---

Ex. 请设计一个输入模块，从键盘依次键入参会成员的姓名，直到输入“#”不再输入，输入的姓名存储在一个列表里

```
lst = []  
name = input("Please input the name:")  
while name != "#":  
    lst.append(name)  
    name = input("Please input the name:")
```

#### 4. 采用递推法计算 $\sin x$ 幂级数展开式的近似值，通项绝对值小于 $10^{-8}$ 时停止计算

思路：

利用右侧的递推式，计算 $\sin x$ 的通项，当通项绝对值小于 $10^{-8}$ 时停止计算

$$\sin x = \frac{x}{1} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\text{通项 } a_n = (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!}$$

$$\text{递推关系 } a_{n+1} = (-1) \frac{x^2}{2n(2n+1)} a_n$$

对于“当xxx时停止计算”或“当xxx时计算”这类事件控制，多用while循环；

对于已知循环次数、循环内容的事件，多用for循环

4. 采用递推法计算sinx幂级数展开式的近似值，通项绝对值小于 $10^{-8}$ 时停止计算

答案：

```
x = float(input())
general_term = x
n = 0
```

$$\text{递推关系 } a_{n+1} = (-1) \frac{x^2}{2n(2n+1)} a_n$$

```
sinx = x
```

```
while abs(general_term) > 1e-8:
```

考察对象是通项的绝对值

```
    n += 1
```

```
    general_term = general_term * (-1) * x * x / ((2 * n) * (2 * n + 1))
```

```
    sinx += general_term
```

```
print("{:.1f}".format(sinx))
```

```
while cond:
```

```
    xxx
```

等价于

```
while True:
```

```
    xxx
```

```
    if not cond:
```

```
        break
```

#### 4. 采用递推法计算sinx幂级数展开式的近似值，通项绝对值小于 $10^{-8}$ 时停止计算

Ex. 计算 $\pi$ 、 $e$ 的值

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \dots + (-1)^{n+1} \frac{1}{2n-1} + \dots$$

```
pi = 0
general_term = 1
n = 1
while abs(general_term) > 10 ** -6:
    n += 1
    pi += general_term
    general_term = (-1)**(n + 1) / (2 * n - 1)
print("Pi = {:.5f}".format(4 * pi))
```

$$\frac{\pi}{2} = 1 + \frac{1}{6} - \dots + \frac{(2n-1)!!}{(2n)!!} \frac{1}{2n+1} + \dots$$

```
pi = 0
general_term = 1
n = 0
while abs(general_term) > 10 ** -10:
    n += 1
    pi += general_term
    general_term = general_term * (2 * n - 1) *
(2 * n - 1) / (2 * n) / (2 * n + 1)
    print(general_term)
print("Pi = {:.2f}".format(2 * pi))
```



#### 4. 采用递推法计算sinx幂级数展开式的近似值，通项绝对值小于 $10^{-8}$ 时停止计算

Ex. 计算 $\pi$ 、 $e$ 的值

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \dots + \frac{1}{n!} + \dots$$

```
e = 1
general_term = 1
n = 0
while abs(general_term) > 10 ** -10:
    n += 1
    general_term = general_term / n
    e += general_term
print("e = {}".format(e))
```

利用递推式可以节省  
运算时间  
解决问题的关键是找  
对递推式

5. 绝对值排序：输入 $n(n \leq 100)$ 个整数，按照绝对值从大到小排序后输出（可能输入多组）

---

**思路：**

输入模块：先让用户输入需要输入几组数据，根据组数进行循环输入

排序模块：利用排序算法（见课件）

输出模块：可以选择每排好一组就输出也可以选择全部排好之后利用循环输出

5. 绝对值排序：输入 $n$ ( $n \leq 100$ )个整数，按照绝对值从大到小排序后输出（可能输入多组）

---

答案：

```
line = int(input())
raw_data = []
for i in range(0,line):
    tmp_data = [int(x) for x in input().split()]
    raw_data.append(tmp_data)
for raw_line in raw_data:
    sorted_line = sorted(raw_line,key=abs,reverse=True)
    sorted_str = [str(x) for x in sorted_line]
    print(' '.join(sorted_str))
```

可替换成任意  
排序算法

5. 绝对值排序：输入n(n≤100)个整数，按照绝对值从大到小排序后输出（可能输入多组）

答案：

```
#输入模块
for raw_line in raw_data:
    sorted_line = sorted(raw_line, key=abs, reverse=True)
#输出模块
```

冒泡排序法

```
n = len(raw_line)
for i in range(n - 1):
    flag = 1
    for j in range(n - 1 - i):
        if abs(raw_line[j]) < abs(raw_line[j + 1]):
            raw_line[j], raw_line[j + 1] = raw_line[j + 1], raw_line[j]
            flag = 0
    if flag:
        break
sorted_str = [str(x) for x in raw_line]
```

如果要交换列表lst中索引为a和b的两个元素

✓: `lst[a], lst[b] = lst[b], lst[a]`

✗: `x = lst[a]; y = lst[b]`


`x, y = y, x` ✗

5. 绝对值排序：输入 $n$ ( $n \leq 100$ )个整数，按照绝对值从大到小排序后输出（可能输入多组）

答案：

```
#输入模块
for raw_line in raw_data:
    sorted_line = sorted(raw_line, key=abs, reverse=True)
#输出模块
```

选择排序法




```
n = len(raw_line)
for i in range(n - 1):
    point = i
    for j in range(i + 1, n):
        if abs(raw_line[point]) < abs(raw_line[j]):
            point = j
    if point != i:
        raw_line[i], raw_line[point] = raw_line[point], raw_line[i]
sorted_str = [str(x) for x in raw_line]
```

5. 绝对值排序：输入 $n$ ( $n \leq 100$ )个整数，按照绝对值从大到小排序后输出（可能输入多组）

答案：

```
#输入模块
for raw_line in raw_data:
    sorted_line = sorted(raw_line, key=abs, reverse=True)
#输出模块
```

插入排序法



```
n = len(raw_line)
for i in range(1, n):
    temp = raw_line[i]
    j = i - 1
    while abs(temp) > abs(raw_line[j]) and j >= 0:
        raw_line[j + 1] = raw_line[j]
        j -= 1
    raw_line[j + 1] = temp
sorted_str = [str(x) for x in raw_line]
```

## 6. 求矩阵的两对角线上的元素之和

**思路：**

输入模块：先让用户输入矩阵的阶数，利用循环输入矩阵

计算模块：寻找对角线上元素索引的规律，根据规律求和。注意区分矩阵是奇数阶还是偶数阶

(0,0)	(0,1)	...	(0,j)	...	(0,n)
(1,0)	(1,1)	...	(1,j)	...	(1,n)
...	...	...	...	...	...
(i,0)	(i,1)	...	(i,j)	...	(i,n)
...	...	...	...	...	...
(n,0)	(n,1)	...	(n,j)	...	(n,n)

— 左上右下对角线：  $x = y$

— 左下右上对角线：  $x + y = n$   
 $n$  是 阶数 - 1

奇数阶两条对角线有交叉元素  
偶数阶两条对角线无交叉元素



## 6. 求矩阵的两对角线上的元素之和

答案：

```
line = int(input())
```

```
matrix = []
```

```
for line_data in range(0,line):
```

```
    matrix_line = [int(i) for i in input().split()]
```

输入第i行

```
    matrix.append(matrix_line)
```

```
sum_1 = 0; sum_2 = 0
```

sum\_1是左上右下和，sum\_2是左下右上和

```
for i in range(0,line):
```

```
    sum_1 += matrix[i][i]
```

```
    sum_2 += matrix[i][line - 1 - i]
```

也可以用matrix[i][-i-1]

```
sum_ = sum_1 + sum_2
```

```
if line % 2 == 1:
```

```
    mid = int((line - 1) / 2)
```

直接进行除运算得到的结果是float，不能用作索引！

```
    sum_ -= matrix[mid][mid]
```

```
print(sum_)
```

## 6. 求矩阵的两对角线上的元素之和

Ex1. 从键盘输入 $n \times n$ 的矩阵，把它转置后输出

```
line = int(input())
matrix = []
for line_data in range(0,line):
    matrix_line = [int(i) for i in input().split()]
    matrix.append(matrix_line)
    for i in range(line):
        for j in range(0,i):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
for matrix_line in matrix:
    for element in matrix_line:
        print(element,end=' ')
    print()
```

只需交换半边，否则又会交换回来

也可写作：  
print(\*matrix\_line)

(0,0)	(0,1)	...	(0,n)
(1,0)	(1,1)	...	(1,n)
...	...	...	...
(n,0)	(n,1)	...	(n,n)

矩阵转置即把 $a[i][j]$ 元素与 $a[j][i]$ 进行调换

## 6. 求矩阵的两对角线上的元素之和

Ex2. 从键盘输入 $n \times n$ 的矩阵，并计算矩阵的鞍点。鞍点是矩阵中的一个位置，该位置上的元素在其所在的行上最大、列上最小（一个矩阵也可以没有鞍点）【假设每行每列内没有相同的元素】

#输入部分与前面相同

saddle = []

for row in range(line):

max\_index = 0

for j in range(len(matrix[row])):

if matrix[row][j] > matrix[row][max\_index]:

max\_index = j

min\_index = 0

for i in range(line):

if matrix[i][max\_index] < matrix[min\_index][max\_index]:

min\_index = i

if min\_index == row:

saddle.append([min\_index,max\_index])

注意：输入的是一个矩阵不一定是方阵

寻找该行的最大元素索引max\_index

寻找max\_index列的最小元素索引min\_index，如果min\_index与row相等则此点是鞍点

## 6. 求矩阵的两对角线上的元素之和

Ex2. 从键盘输入 $n \times n$ 的矩阵，并计算矩阵的鞍点。鞍点是矩阵中的一个位置，该位置上的元素在其所在的行上最大、列上最小（一个矩阵也可以没有鞍点）

(接上)

```
if saddle:
```

```
    for saddle_point in saddle:
```

```
        print(saddle_point)
```

```
else:
```

```
    print("Not found!")
```

一个矩阵可以没有鞍点

2	4	5	1
1	3	2	0
6	5	7	3

鞍点为[1,1]

## 6. 求矩阵的两对角线上的元素之和

Ex2. 从键盘输入 $n \times n$ 的矩阵，并计算矩阵的鞍点。鞍点是矩阵中的一个位置，该位置上的元素在其所在的行上最大、列上最小（一个矩阵也可以没有鞍点）

#输入部分同上

```
row_max = []
for row in range(line):
    max_index = 0
    for j in range(len(matrix[row])):
        if matrix[row][j] > matrix[row][max_index]:
            max_index = j
    row_max.append([row, max_index])
col_min = []
for col in range(len(matrix[0])):
    min_index = 0
    for i in range(line):
        if matrix[i][col] < matrix[min_index][col]:
            min_index = i
    col_min.append([min_index, col])
```

先求每一行内最大值的位置  
存在row\_max中

再求每一列内最小值的位置  
存在col\_min中

## 6. 求矩阵的两对角线上的元素之和

Ex2. 从键盘输入 $n \times n$ 的矩阵，并计算矩阵的鞍点。鞍点是矩阵中的一个位置，该位置上的元素在其所在的行上最大、列上最小（一个矩阵也可以没有鞍点）

```
saddle = []  
for coord in row_max:  
    if coord in col_min:  
        saddle.append(coord)  
if saddle:  
    for saddle_point in saddle:  
        print(saddle_point)  
else:  
    print("Not found!")
```

如果一个坐标同时出现在row\_max和col\_min中，说明这个坐标是鞍点

(0,1)	2	4	5	1
(1,1)	1	3	2	0
(2,0)	6	5	7	3
	(1,0)	(1,1)	(1,2)	(1,3)

## 6. 求矩阵的两对角线上的元素之和

Ex3. 生成3×4矩阵，并输出

```
matrix = []
value = 1
for i in range(3):
    matrix_line = []
    for j in range(4):
        matrix_line.append(value)
        value += 1
    matrix.append(matrix_line)
for i in range(3):
    for j in range(4):
        print(matrix[i][j],end=" ")
    print()
```

1	2	3	4
5	6	7	8
9	10	11	12

依次随着行、列的增加，  
矩阵的元素依次加一

对于次序比较规律矩阵  
生成，可以直接利用  
append生成矩阵



## 6. 求矩阵的两对角线上的元素之和

Ex4. 生成n阶方阵，n从键盘输入，如n=4时如右图

```
n = int(input("Input the dimension of the matrix: "))
```

```
matrix = []  
for i in range(n):  
    matrix.append([0] * n)  
value = 1
```

```
for i in range(n):  
    if i % 2 == 0:  
        for j in range(n):  
            matrix[i][j] = value  
            value += 1  
    else:  
        for j in range(1, n + 1):  
            matrix[i][-j] = value  
            value += 1
```

#输出部分参考Ex3

注意：不能用`matrix = [[0] * n] * n`  
对`matrix`进行初始化

直接利用`[0] * n`生成一维列表  
没有问题；但是利用一维列表  
乘法生成二维列表时，由于一  
维列表的id相同，因此更改某  
一个列表会对所有的列表进行  
更改

方法不唯一！

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

In[1]: `matrix = [[0] * 3] * 2`

In[2]: `id(matrix[0]) == id(matrix[1])`  
Out[2]: True

对于索引变化有规律但是不是依  
次变化时，可以先对矩阵进行初  
始化再根据规律对对应位置上的  
值进行更改

# 6. 求矩阵的两对角线上的元素之和

Ex5. 生成n阶方阵， n从键盘输入， 如n=4时如右图

```
#输入及初始化部分同Ex4
value = 1; i = 0; j = 0
flag = 1
while True:
    matrix[i][j] = value
    value += 1
    ischange = 0
    if flag % 4 == 1:
        xxxxxx
    if flag % 4 == 2:
        xxxxxx
    if flag % 4 == 3:
        xxxxxx
    if flag % 4 == 0:
        xxxxxx
```

根据规律不断更改下一个元素的索引

见下页

```
if flag % 4 == 1:
    xxxxxx
if ischange == 0:
    break
#输出部分同Ex4
```

见后页

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

flag代表矩阵增加的“方向”， 根据规律矩阵中的元素依次向右、下、左、上增加。当flag % 4 == 1时代表向右， 随着余数的改变， 方向依次变化。

ischange用来监测矩阵是否能继续“增加”， 如果向四个方向尝试后均失败则代表已经到达最后一个元素， 跳出循环。由于已知最后一个元素的值， 因此也可以利用最后一个值对循环进行控制

## 6. 求矩阵的两对角线上的元素之和

Ex5. 生成n阶方阵，n从键盘输入，如n=4时如右图

```
if flag % 4 == 1:
    if j < n-1:
        if matrix[i][j+1] == 0:
            j += 1; ischange = 1
        continue
    flag += 1
```

向右的条件：  
① 没有到矩阵最右侧  
② 右侧的元素没有被改变过（仍是初始化值）  
剩下3个同理

```
if flag % 4 == 3:
    if j > 0:
        if matrix[i][j-1] == 0:
            j -= 1; ischange = 1
        continue
    flag += 1
```

向左

```
if flag % 4 == 2:
    if i < n-1:
        if matrix[i+1][j] == 0:
            i += 1; ischange = 1
        continue
    flag += 1
```

向下

```
if flag % 4 == 0:
    if i > 0:
        if matrix[i-1][j] == 0:
            i -= 1; ischange = 1
        continue
    flag += 1
```

向上

## 6. 求矩阵的两对角线上的元素之和

Ex5. 生成n阶方阵，n从键盘输入，如n=4时如右图

```
if flag % 4 == 1:
    xxxxxx
if ischange == 0:
    break
```

#输出部分同Ex4

在从向上到向右转时，如果不加这一句经过四个判断矩阵的值都没有改变，使得ischange不变导致跳出循环

改成if value ==  $n^2 + 1$ 也可，当value到最大时直接跳出循环。

**!** 这时也不能删掉上面的if flag % 4 == 1代码，否则也会有问题

方法不唯一!

## 6. 求矩阵的两对角线上的元素之和

Ex6. 生成n阶蛇形矩阵，n从键盘输入，如n=4时如右图

```
n = int(input("Input the dimension of the matrix: "))
matrix = []
for i in range(n):
    matrix.append([0] * (n - i))
value = 1
for sum_ in range(n):
    for j in range(sum_ + 1):
        i = sum_ - j
        matrix[i][j] = value
        value += 1
for i in range(n):
    for j in range(len(matrix[i])):
        print(matrix[i][j], end=" ")
    print()
```

初始化matrix

利用右侧规律  
生成矩阵

1	3	6	10
2	5	9	
4	8		
7			

	(0,0)	(0,1)	(0,2)	(0,3)
0	(1,0)	(1,1)	(1,2)	(1,3)
1	(2,0)	(2,1)	(2,2)	(2,3)
2	(3,0)	(3,1)	(3,2)	(3,3)
3	4	5	6	

左下右上索引的和是固定值  
左上右下索引的差是固定值

方法不唯一!

# 6. 求矩阵的两对角线上的元素之和

Ex7. 生成n阶蛇形矩阵，n从键盘输入，如n=4时如下图  
(Ex7 = Ex6 + o(Ex6))

1	2	4	7
3	5	8	
6	9		
10			

Ex8. 生成n阶蛇形矩阵，n从键盘输入，如n=4时如下图  
(Ex7 = Ex6 + Ex4)

1	3	4	10
2	5	9	
6	8		
7			

Ex9. 生成n阶蛇形矩阵，n从键盘输入，如n=4时如下图  
(Ex7 = Ex6 + Ex5)

1	2	3	4
9	10	5	
8	6		
7			

Ex10. 生成n阶蛇形矩阵，n从键盘输入，如n=4时如下图  
(Ex7 = Ex6 + Ex6)

1			
5	2		
8	6	3	
10	9	7	4

大部分有规律的矩阵都可以用来练习矩阵问题多转化为找索引规律的问题

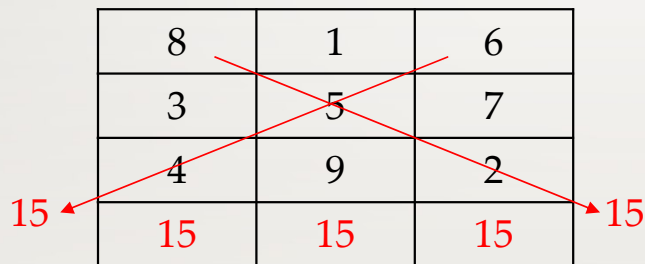


## 6. 求矩阵的两对角线上的元素之和

Ex11. 生成 $n$ 阶 ( $n$ 为奇数) 幻方, 填入 $1, 2, \dots, N^2$ , 使得每行、每列以及两个对角线上的数的和均相等, 并对结果进行验证 (即求每行、每列以及两条对角线的数的和)。

**提示:** 把1填到第一行最中间的格子中, 剩下的数按照如下方法填写。如果当前格子是方阵中最右上角的格子, 则把下一个数填在下一行的同一列格子中; 否则, 如果当前格子在第一行上, 则把下一个数填在下一列的最后一个格子中; 否则, 如果当前格子在最后一列上, 则把下一个数填在上一行的第一列格子中; 否则, 如果当前格子的右上角格子里没有数, 则在其中填入下一个数, 否则把下一个数填在下一行的同一列格子中。

8	1	6
3	5	7
4	9	2
15	15	15



三阶幻方



## 6. 求矩阵的两对角线上的元素之和

Ex11. 生成n阶 (n为奇数) 幻方

```
#初始化同Ex4
value = 1
i = 0; j = int((n - 1) / 2)
while value <= n ** 2:
    matrix[i][j] = value
    value += 1
    if i == 0 and j == n-1:
        i += 1
        continue
    if i == 0:
        j += 1; i = n-1
        continue
    if j == n-1:
        i -= 1; j = 0
        continue
```

所有的if都可以改成elif, 此时  
可以把所有的continue删掉

```
if matrix[i-1][j+1] == 0:
    i = i - 1; j = j + 1
    continue
else:
    i += 1
```

#输出部分同Ex4

根据提示依次填入数即可

## 6. 求矩阵的两对角线上的元素之和

Ex11. 生成n阶 (n为奇数) 幻方

```
row_sum = []  
for row in matrix:  
    row_sum.append(sum(row))  
print(row_sum)
```

如果只是关注可迭代对象内的元素内容而不关注其索引，可以直接用  
`for x in iterable`，没必要都用`for i in range(len(iterable))`  
下面求每一列的和时，因为需要利用索引求所以用`for j in range(n)`求解，而求每一行的和时，不需要索引也可以把每一行的元素提取出来

```
col_sum = []  
for j in range(n):  
    sum_ = 0  
    for i in range(n):  
        sum_ += matrix[i][j]  
    col_sum.append(sum_)  
print(col_sum)
```

注意顺序!

```
sum_1 = 0; sum_2 = 0  
for i in range(n):  
    sum_1 += matrix[i][i]  
    sum_2 += matrix[i][n-1-i]  
print(sum_1, sum_2)
```

两对角线和

## 6. 求矩阵的两对角线上的元素之和

Ex12\*. 生成n阶 (n为偶数) 幻方

**提示：** 如果n为4的倍数，采用对称元素交换法：

- ① 把数1到 $n^2$ 按从上到下，从左到右的顺序填入矩阵（类似Ex3）
- ② 将所有 $4 \times 4$ 子方阵的两对角线上位置的数不变，其他位置上的数关于方阵中心作对称变换，即 $a[i][j]$ 与 $a[n-1-i][n-1-j]$

如果n是其他偶数时，即n可以表示成 $4m + 2$ 的形式时

- ① 首先把大方阵分解成4个奇数 ( $2m + 1$ 阶) 子方阵。
- ② 按Ex11中奇数幻方给分解的4个子方阵对应赋值，由小到大依次为上左子阵 (i)，下右子阵 ( $i + v$ )，上右子阵 ( $i + 2v$ ) 和下左子阵 ( $i + 3v$ )，即4个子方阵对应元素相差v，其中 $v = n^2 / 4$
- ③ 然后作相应的元素交换： $a[i][j]$ 与 $a[i+u][j]$ 在同一列做对应交换 ( $j < t$ 或 $j > n-t+2$ )， $a[t-1][0]$ 与 $a[t+u-1][0]$ 、 $a[t-1][t-1]$ 与 $a[t+u-1][t-1]$ 两对元素交换，其中 $u = n / 2$ ， $t = (n + 2) / 4$ ，上述交换使行列及对角线上元素之和相等。

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



1	15	14	4
12	6	7	9
8	10	11	5
13	3	2	16

## 6. 求矩阵的两对角线上的元素之和

Ex13. 利用代码实现矩阵乘法（假设所有输入均满足矩阵乘法）

设  $A = (a_{ij})_{m \times n}$ ,  $B = (b_{ij})_{n \times l}$ ;

若  $C = A \times B$ , 则  $C = (c_{ij})_{m \times l}$ , 其中  $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$

```
matrix_a = []; matrix_b = []
matrix_line = input("Please input the matrix A:\n")
while matrix_line:
    matrix_a.append([float(x) for x in matrix_line.split()])
    matrix_line = input()

matrix_line = input("Please input the matrix B:\n")
while matrix_line:
    matrix_b.append([float(x) for x in matrix_line.split()])
    matrix_line = input()
```

输入模块：如果直接回车视为输入完成

```
matrix_c = []
for i in range(len(matrix_a)):
    matrix_line = []
    for j in range(len(matrix_b[0])):
        c = 0
        for k in range(len(matrix_b)):
            c += matrix_a[i][k] * matrix_b[k][j]
        matrix_line.append(c)
    matrix_c.append(matrix_line)
```

计算矩阵C

```
for i in range(len(matrix_c)):
    for j in range(len(matrix_c[0])):
        print(matrix_c[i][j], end=' ')
    print()
```

输出

7. 统计单词词频：从键盘输入一个英文句子，除单词和空白字符外句子中只包含“、”、“.”、“’”、“””和“!”几个标点符号，请统计词频（全部转为小写）

答案：

```
sentence = input().lower()
freq_dict = {}
com_str = ",.\'\"!"
for com in com_str:
    sentence = sentence.replace(com, ' ')
sentence = sentence.strip()
sentence_lst = sentence.split()
```

```
for char in sentence_lst:
    if char in freq_dict:
        freq_dict[char] += 1
    else:
        freq_dict[char] = 1
```

```
sorted_lst = sorted(freq_dict.items(), key=lambda x: (x[1], x[0]))
```

```
for item in sorted_lst:
    print(item[0], item[1])
```

replace方法会把字符串中的所有待替换的子字符串替换！其返回值为替换的字符串

根据已有的列表或输入构建词典的方法！  
一定要理解！

sorted函数见实验3

7. 统计单词词频：从键盘输入一个英文句子，除单词和空白字符外句子中只包含“,”、“.”、“'”、“””和“!”几个标点符号，请统计词频（全部转为小写）

**不要对循环中的可迭代对象进行操作！（包括但不限于删除元素、增加元素）**

```
test_lst = [1,2,2,3,3,4,4,4,4,5,6,6,7]
for num in test_lst:
    if num % 2 == 0:
        test_lst.remove(num)
```

运行此段代码后，test\_lst并不会把所有的偶数去掉，最终结果为[1, 2, 3, 3, 4, 4, 5, 6, 7]

```
test_lst = [1,2,3]
index = 3
for num in test_lst:
    print(num,end=' ')
    index += 1
    test_lst.append(index)
    if index > 8:
        break
```

此循环体会运行6遍，而并非3遍。  
输出结果为：1 2 3 4 5 6

具体原理与迭代器（iterator）有关，不明白也可以。  
只需要注意不要对循环中的对象进行更改即可