# PS1_1 根据流程图比较三个随机数的大小

首先导入随机数，并且打印出来，这里我打印出 a，b，c，三个随机整数。

```
import random
a=random.randint(0,100)
b=random.randint(0,100)
c=random.randint(0,100)
print(a,b,c)
```

第二，定义了一个函数，函数内容为比较这三个随机数的大小（有六种情况）：

第一步先比较 a 和 b 的大小，如果成立，继续比较 b 和 c 的大小，若 b>c 成立，则得到“情况 1：a>b>c”；若 b>c 不成立，（此时大小：a>b 且 c>b）继续比较 a 和 c，若 a>c 成立，则得到“情况 2：a>c>b”，若 a>c 不成立，则得到“情况 3：c>a>b”。

```
    if a>b:
        if b>c:
            print(a,b,c)
        else:
            if a>c:
                print(a,c,b)
            else:
                print(c,a,b)
```

到这里已经把 a>b 成立的所有情况考虑进去，第二步应考虑 a>b 不成立的情况：

第二步：b>a，比较 b 和 c 大小，若 b>c 成立，此时大小：b>a 且 b>c，那么应该在比较 a 和 c 的大小，若 a>c 成立，则得到“情况 4：b>a>c”；若 a>c 不成立，则得到“情况 5：b>c>a”；若一开始的 b>c 不成立，则得到“情况 6：c>b>a”。

至此已经考虑了所有大小情况，最后将函数调用起来即可。

```
    else:
        if b>c:
            if a>c:
                print(b,a,c)
            else:
                print(b,c,a)
        else:
            print(c,b,a)
value()
```

完整脚本如下：

```
@author: WANG
"""
import random
a=random.randint(0,100)
b=random.randint(0,100)
c=random.randint(0,100)
print(a,b,c)
def value():
    if a>b:
        if b>c:
            print(a,b,c)
        else:
            if a>c:
                print(a,c,b)
            else:
                print(c,a,b)
    else:
        if b>c:
            if a>c:
                print(b,a,c)
            else:
                print(b,c,a)
        else:
            print(c,b,a)
value()
```

# PS1_2

一、生成随机矩阵 5*10　10*5

首先将 numpy（Numberical Python）包导入，并命名为 np，因为 numpy 中已包含 random，因此 random 已经可以使用，无需另外导入。

接着使用 np.random.randint(low,hight,size,dtype)函数，生成一个整数或 N 维整数数组。在此我们要求矩阵 M1 为 5*10 矩阵，其中元素为 0-50 间的随机整数；M2 为 10*5 矩阵，其中元素为 0-50 的随机整数。

```
import numpy as np

M1=np.random.randint(0,51,size=(5,10))
M2=np.random.randint(0,51,size=(10,5))

print(M1,M2)
```

二、将生成的两个矩阵进行矩阵乘法运算

首先，矩阵乘法得到的第 i 行第 j 列的元素：M1 第 i 行的所有元素与 M2 第 j 列的所有元素的乘积和。例如：Result 中的第一行第一列的元素：是 M1 中第一行元素与 M2 中第一列元素的乘积和，要想得到 Result 所有第一行的元素（5 个元素），就要 M1 中第一行元素分别与 M2 的每一列分别求乘积和。那么总的来说，想要得到 Result 这个 5 行 5 列的矩阵，就需要进行 25 次乘积和，M1 的第一行所有元素先与 M2 每一列所有元素求乘积和（即做了 5 次乘积和），接着是第二行、第三行……如此循环 5 次。

因此需要将 M1 的每一行（M2 的每一列）做循环嵌套。

```
for i in range(len(M1)):#01234
    for j in range(len(M2[0])):#01234
        for k in range(len(M2)):#0123456789
```

这里 i 表示能取 01234 这 5 个数字，代表了整个大循环需要循环 5 次，接着是 j 能取到 01234 这五个数字，表示了需要对 M2 的列遍历 5 次，K 能取到 0123456789 这十个数字，代表了 M1 列数以及 M2 的行数。那么 M1 和 M2 中的各个元素就能表示为 M1[i][k]，M2[k][j]。

将 Result 定义为一个 5 行 5 列的零矩阵：`result = np.zeros((5,5),int)`

表达式就为：`result[i][j] += M1[i][k] * M2[k][j]`

完整脚本如下

```
Created on Sun Oct  3 19:42:05 2021

@author: WANG
"""
#from random import randint
#M=random.randint(0,50,(5,10))

import numpy as np

M1=np.random.randint(0,51,size=(5,10))
M2=np.random.randint(0,51,size=(10,5))

print(M1,M2)


result = np.zeros((5,5),int)
for i in range(len(M1)):#01234
    for j in range(len(M2[0])):#01234
       for k in range(len(M2)):#0123456789
           result[i][j] += M1[i][k] * M2[k][j]
           #print(result[i][j])

print(result)
```

（参考网上资料：https://cloud.tencent.com/developer/article/1351435）

# PS1_3 打印帕斯卡三角形的某一行

因帕斯卡三角形前两行（1 行和 2 行）较为简单，先用列表的形式储存起来：`result=[[1],[1,1]]`，之后再利用 append 将后面的行加上去。我们想只打印出某一行时，只需要利用列表索引，就能将它定位并且打印出来。

那么怎么表示现在怎么表示出某一行的数字呢？由于帕斯卡三角形的特点：第 k 行的第 i（i 不取头尾位置）个数字=第 k-1 行的第 i-1 和 i 的加和。对此仍然可以利用列表的 id 位置来表示。对于头尾的两个数字，则无论是哪一行都是 1，那么我们只需要在求出中间那些数字之后，再在一前一后各加上一个 1。

```
result=[[1],[1,1]]
a=[1,1]
for i in range(2,k):
    r=[]
    for j in range(len(a)-1):
        r.append(a[j]+a[j+1])
```

由于 result=[[1],[1,1]]，对于 result 列表来说，想要表达出后面行的数字列表的话，需要根据 a=[1,1]这个列表，因此单独取出来。那么第 3 行的的头尾两个数字我们已经知道都是 1，而第二位的数字等于 a[0]和 a[1]的和，即得到 2，我们将这个数字存在 r 这个空列表中，接着在 r 的前面和后面各合并一个仅有 1 的列表：[1]，那么我们就得到了第 3 行的数字列表。由此来看，要表达第 k 行，就得增加 k-2 个数字到空列表 r 中，因此引入一个 for 循环，取 i 在[2,k)范围。为了表达具体的数字，还需在后面再引入 for 循环，计算具体的数字，循环行数-2 次（由于行数等于行内数字个数，加上 range 取不到最后一位，因此取 j 在[0,len(a))范围内）。在此之后将[1]+r+[1]赋值给 a 列表，再将 a 列表添加到 result 的最后一位。

```
result=[[1],[1,1]]
a=[1,1]

for i in range(2,k):
    r=[]
    for j in range(len(a)-1):
        r.append(a[j]+a[j+1])
    a=[1]+r+[1]
    result.append(a)
```

最后我们得到的 result 列表中实际上储存了 k 行的全部数值，但由于我们仅需要第 k 行的，因此，只需打印 result[k-1]即可（在这里有一个差值，由于列表索引起始于 0，而正常计算行数起始于 1，因此打印的是 k-1 位而不是 k 位，需要注意。）。

全部脚本如下：

```
def Pascal_triangle(k):
    result=[[1],[1,1]]
    a=[1,1]

    for i in range(2,k):
        r=[]
        for j in range(len(a)-1):
            r.append(a[j]+a[j+1])
        a=[1]+r+[1]
        result.append(a)
    #print(result)
    print(result[k-1])
Pascal_triangle(100)
Pascal_triangle(200)
```

# PS1_4 计算步伐数（加一或双倍）

这个问题可以反向思考，当给定一个数 a（a>1）时，我们时而减 1，时而除以 2，直到得到 1。在 a>1 的情况下，我们只需要判断 a 这个数是不是偶数，如果是偶数就除以 2（当然我们也能通过减 1 来达到同样目的，但是这样就慢了，所以我们要尽可能的多使用除以 2），如果不是偶数就减去 1，循环直到 a==1，设置 i 来计算步伐数量，期间每实施一次除法或减法，i 都要加上 1。

完整脚本如下：

```
def Least_moves(a):
    i=0
    while a > 1:
        if a % 2 ==0:
            a=a/2
        else:
            a=a-1
        i+=1
    print(i)
Least_moves(10)
```

# PS1_5

## 5.1（左小幸同学向我解释了此题）

_1_2_3_4_5_6_7_8_9，在每个横线上（除了第一个），每个横线上都能添加三种符号：“+”、“-”以及“”。就能建立一个符号的列表，然后用 for 循环进项遍历，共遍历 8 次；第一条横线只有两种情况：“-”或“”，因为加号和不加东西的情况是一样的。因此我们就能写出这样的循环：

```
a='123456789'
b=['+','-','']
c=['-','']

for i1 in b:
    for i2 in b:
        for i3 in b:
            for i4 in b:
                for i5 in b:
                    for i6 in b:
                        for i7 in b:
                            for i8 in b:
                                for i9 in c:
```

接着利用索引，将符号与数字连接在一起，并将表达式结果算出来：

```
for i9 in c:
    result=i9+a[0]+i1+a[1]+i2+a[2]+i3+a[3]+i4+a[4]+i5+a[5]+i6+a[6]+i7+a[7]+i8+a[8]
    er=eval(result)
```

最后取一个 1-100 的随机整数 A，若计算结果 er 与 A 相等，就将表达式 result 与结果 er 打印出来。d 用于计算整数 A 有几种表示法：

```
    result=i9+a[0]+i1+a[1]+i2+a[
    er=eval(result)
    if er == A:
        d+=1
        print(result,'=',er)
```

## 5.2

如果算出来的数值 er 在 1-100 范围内，那么把 er 储存到空列表 T 中，利用 count()函数，能将 T 中的某个数字 i 的次数计算出来，并将之添加到 Total_solutions 的空列表中，此时，数字 i 与 index+1 是相同的。因此我们就能利用 max 和 min 函数找到最多的解决方案，并且利用 index 找到该数字。

```
                                    if er in range(1,101):
                                        T.append(er)
print(A,"的解决方案有",d,"种")




Total_solutions = []
for i in range(1,101):

    Total_solutions.append(T.count(i))
    max_solution=max(Total_solutions)
    min_solution=min(Total_solutions)

    max_num = Total_solutions.index(max_solution)+1
    min_num = Total_solutions.index(min_solution)+1

print('0-100中：',max_num,'产生的解决方案最多，共有',max(Total_solutions),'种')
print('0-100中：',min_num,'产生的解决方案最少，共有',min(Total_solutions),'种')
```

完整的脚本如下：

```
import random
A=random.randint(0,100)
print(A)
a='123456789'
b=['+','-','']
c=['-','']
d=0
T=[]
for i1 in b:
    for i2 in b:
        for i3 in b:
            for i4 in b:
                for i5 in b:
                    for i6 in b:
                        for i7 in b:
                            for i8 in b:
                                for i9 in c:
                                    result=i9+a[0]+i1+a[1]+i2+a[2]+i3+a[3]+i4+a[4]+i5+a[5]+i6+a[6]+i7+a[7]+i8+a[8]
                                    er=eval(result)
                                    if er == A:
                                        d+=1
                                        print(result,'=',er)
                                    if er in range(1,101):
                                        T.append(er)
print(A,"的解决方案有",d,"种")
Total_solutions = []
for i in range(1,101):

    Total_solutions.append(T.count(i))
    max_solution=max(Total_solutions)
    min_solution=min(Total_solutions)
    max_num = Total_solutions.index(max_solution)+1
    min_num = Total_solutions.index(min_solution)+1
print('0-100中：',max_num,'产生的解决方案最多，共有',max(Total_solutions),'种')
print('0-100中：',min_num,'产生的解决方案最少，共有',min(Total_solutions),'种')
```

参考了 count 函数的用法：https://www.runoob.com/python/att-string-count.html