In [116…

```
#1.Flowchart
# define a function
def Print_values():
    # To input something
    a = int(input("a="))
    b = int(input("b="))
    c = int(input("c="))
    # List each condition one by one
    if a > b and b > c:
        print(a,b,c)
        # I got a inspiration from f-string(The result is converted to a string
        print(f"{a + b - 10 * c}")
    elif a > b and b < c and a > c:
        print(a,c,b)
        print(f"{a + c - 10 * b}")
    elif a > b and b < c and a < c:
        print(c,a,b)
        print(f"{c + a - 10 * b}")
    elif a < b and b < c:
        print(c,b,a)
        print(f"{c + b - 10 * a}")
    else:
        pass
# Call the function
Print_values()
```

```
10 5 1
5
```

In [118…

```
#2.Continuous ceiling function
# To input something.
N = int(input("N = "))
#To use the math,ceil function
import math
def F(x):
    if x == 1:
        return 1
    else:
        return F(math.ceil(x / 3)) + 2 * x
#Convert N to an iterable object
n = [F(x) for x in range(1, N+1)]
print(n)
```

```
[1, 5, 7, 13, 15]
```

In [295…

```
#3.1 借鉴https://blog.csdn.net/K346K346/article/details/50988681中所采用的动态规
x = int(input("骰子的和x = "))
n = 10
def Find_number_of_ways(n, x):
    #dp表达达到和的数量
    dp = [0] * (x + 1)
    #0个骰子，只有一种方法可以得到0
    dp[0] = 1
    #每一个骰子都投一遍
    for i in range(n):
        #从后往前更新dp数组
        for j in range(x, 0, -1):
            #保存目前的dp[j]的值
            temp = dp[j]
```

```
            #经历每一个骰子的面
            for face in range(1, 7):
                if j - face >= 0:
                    #更新方法数
                    temp += dp[j - face]
            #更新现在和的方法数
            dp[j] = temp
    return dp[x]
ways = Find_number_of_ways(n, x)
print(ways)
```

```
1
```

In [284…

```
#3.2
def Find_number_of_ways(n, x):
    dp = [0] * (x + 1)
    dp[0] = 1
    for i in range(n):
        for j in range(x, 0, -1):
            temp = dp[j]
            for face in range(1, 7):
                if j - face >= 0:
                    temp += dp[j - face]
            dp[j] = temp
    return dp
def main():
    n = 10
    max_sum = 60
    min_sum = 10

    dp = Find_number_of_ways(n, max_sum)
    #列出10到60所有的方法数量
    Number_of_ways = dp[min_sum:max_sum+1]
    #找到最大的方法数
    max_ways = max(Number_of_ways)
    #最大方法数的和
    max_sum_index = Number_of_ways.index(max_ways) + min_sum
    print(f"最大方法数是 {max_sum_index} 他的方法有 {max_ways}.")
#感谢师兄对错误的指出，确保main函数只调用1次，如果没有这句代码，会导致无限递归。
if __name__ == "__main__":
    main()
```

```
最大方法数是 30 他的方法有 17538157.
```

In [253…

```
#4.1 Dynamic programming
#To input something
N = int(input("N = "))
import random
def Random_integer(N):
    #I got a inspiration from random.randint(a, b)(can generate a random integer
    return [random.randint(0,10) for i in range(N)]
print(Random_integer(N))
```

```
[4, 8]
```

In [127…

```
#4.2
N = int(input("N = "))
import random
import numpy as np
#got a inspiration from combinations
from itertools import combinations
```

```
#Use the code of 3.1
def Random_integer(N):
    return [random.randint(0,10) for i in range(N)]
print(Random_integer(N))

def Sum_averages(arr):
    #转化为numpy数组
    arr = np.array(arr)
    #Initial summation
    sum = 0.0
    for n in range(1,len(arr) + 1):
       for m in combinations(arr,n):
        #np.mean calculates the average of all elements in an input array or mat
        average = np.mean(m)
        #Cumulative mean
        sum += average
    return sum
random_array = Random_integer(N)
result = Sum_averages(random_array)
print(Sum_averages(random_array))
```

```
[1, 10, 3, 9]
127.49999999999999
```

In [293…

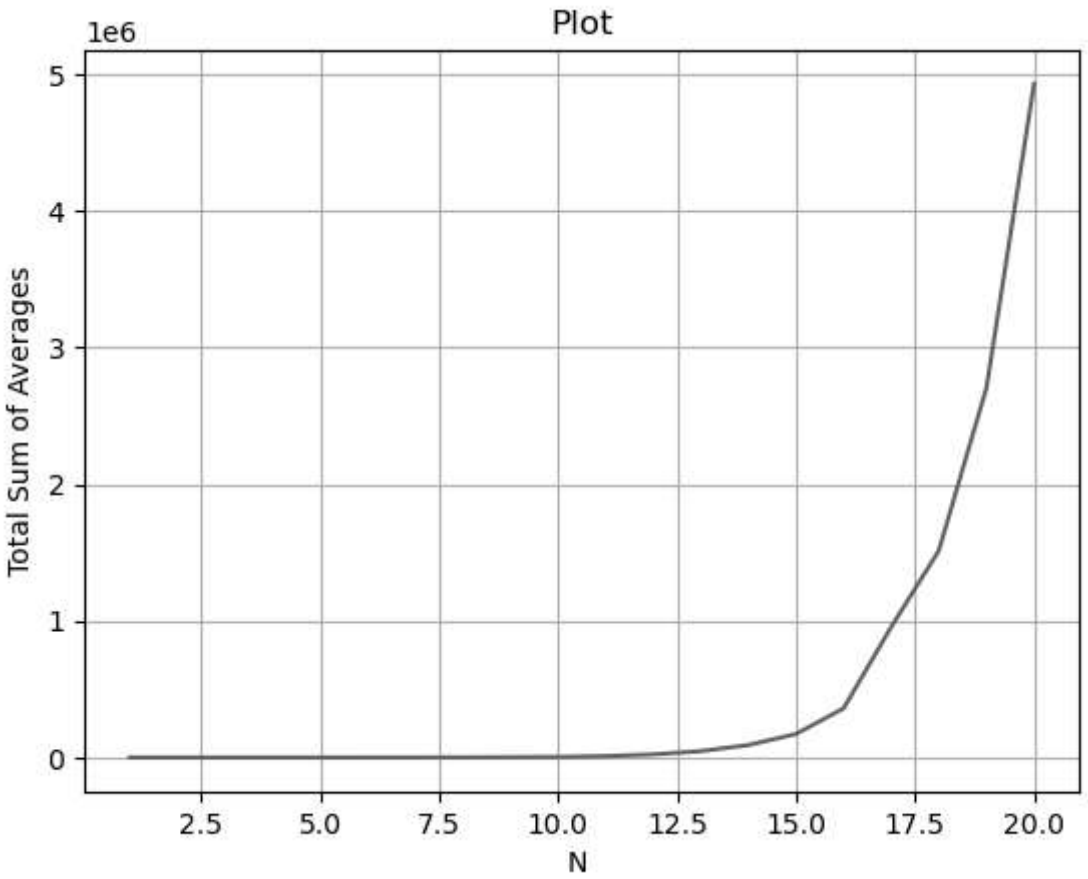
```
#4.3
N = int(input("N = "))
import random
import numpy as np
from itertools import combinations
#To draw graphs
#4.2步骤
import matplotlib.pyplot as plt
def Random_integer(N):
    return [random.randint(0,10) for i in range(N)]
print(Random_integer(N))

def Sum_averages(arr):
    arr = np.array(arr)
    T_sum = 0.0
    for n in range(1, len(arr) + 1):
        for m in combinations(arr, n):
            average = np.mean(m)
            T_sum += average
    return T_sum
#计算1到100的平均值

Total_sum_averages = []
for N in range(1, 21):
    random_array = Random_integer(N)
    result = Sum_averages(random_array)
    #将result添加到Total_sum_average中
    Total_sum_averages.append(result)
#计算量太大，图一直不显示，因此将100的范围改成了20次
plt.plot(range(1, 21), Total_sum_averages)
plt.title("Plot")
#X轴名称
plt.xlabel("N ")
#Y轴名称
plt.ylabel("Total Sum of Averages")
```

```
#添加网格线
plt.grid()
#显示图片窗口
plt.show()
```

```
[7, 3]
```



In [198…

```
#5.1
import numpy as np
N = int(input("N = "))
M = int(input("M = "))
def Create_Matrix(N,M):
    #矩阵中所有数都为0,且矩阵中所有数为整数
    arr1 = np.zeros((N,M),dtype = int)
    #将左上角和右下角的数填充为1
    arr1[0,0] = 1
    arr1[N-1,M-1] = 1

    for i in range(N):
        for j in range(M):
            #将除左上角和右下角的地方随机化
            if (i,j) != (0,0) and (i,j) != (N-1,M-1):
                arr1[i,j] = np.random.randint(0,2)
    return arr1
arr1 = Create_Matrix(N,M)
print(arr1)
```

```
[[1 1 1 1 1]
 [1 0 0 0 1]
 [1 0 0 0 1]
 [1 1 0 0 1]]
```

In [220…

```
#5.2
import numpy as np
```

```
N = int(input("N = "))
M = int(input("M = "))
#以下内容为5.1
def Create_Matrix(N,M):
    arr1 = np.zeros((N,M),dtype = int)
    arr1[0,0] = 1
    arr1[N-1,M-1] = 1

    for i in range(N):
        for j in range(M):
            if (i,j) != (0,0) and (i,j) != (N-1,M-1):
                arr1[i,j] = np.random.randint(0,2)
    return arr1

#参考以下链接：https://blog.csdn.net/qq_29681777/article/details/83719680
def arr1_paths(arr1,x,y):
   #限制边界
    if x < 0 or y < 0 or x >= len(arr1) or y >= len(arr1[0]) or arr1[x][y] == 0:
        return 0
    #到达终点条件
    if x == len(arr1) - 1 and y == len(arr1[0]) - 1:
        return 1
    #向右或向下移动
    right_paths = arr1_paths(arr1, x, y + 1)
    down_paths = arr1_paths(arr1, x + 1, y)
    #返回总路径
    return right_paths + down_paths

def count_path(arr1):
    #检查起点与终点是否可以到达
    if arr1.size == 0 or arr1[0][0] == 0 or arr1[-1][-1] == 0:
       #如果不可以，返回 0
        return 0
    #再从（0,0）开始计数
    return arr1_paths(arr1, 0, 0)

arr1 = Create_Matrix(N,M)
print(arr1)

result = count_path(arr1)
print(result)
```

```
[[1 1 0 0 0 1 0 1]
 [1 0 1 0 0 1 0 1]
 [1 1 1 1 1 0 1 0]
 [1 0 0 0 0 1 1 1]
 [1 1 0 0 0 0 1 1]
 [1 1 1 0 0 0 1 1]]
0
```

In [225…

```
#5.3
#以下为5.2的程序
import numpy as np
def Create_Matrix(N,M):
    arr1 = np.zeros((N,M),dtype = int)
    arr1[0,0] = 1
    arr1[N-1,M-1] = 1
    for i in range(N):
        for j in range(M):
            if (i,j) != (0,0) and (i,j) != (N-1,M-1):
```

```
                arr1[i,j] = np.random.randint(0,2)
    return arr1
def arr1_paths(arr1,x,y):
    if x < 0 or y < 0 or x >= len(arr1) or y >= len(arr1[0]) or arr1[x][y] == 0:
        return 0
    if x == len(arr1) - 1 and y == len(arr1[0]) - 1:
        return 1
    right_paths = arr1_paths(arr1, x, y + 1)
    down_paths = arr1_paths(arr1, x + 1, y)
    return right_paths + down_paths

def count_path(arr1):
    if arr1.size == 0 or arr1[0][0] == 0 or arr1[-1][-1] == 0:
        return 0
    return arr1_paths(arr1, 0, 0)

def main():
    N = 10
    M = 8
    #将路径总数初始化
    total_paths = 0
    #运行1000次
    runs = 1000
    #进行1000次的运算
    for a in range(runs):
        #生成新的矩阵
        arr1 = Create_Matrix(N,M)
        #计算路径并累加
        total_paths += count_path(arr1)
        #计算平均路径数
        mean_paths = total_paths / runs
        print(mean_paths)
main()
```

0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016

0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016

0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016

0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016

0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.016
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286

0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.286
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288

0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.288
0.294
0.294
0.294
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298

0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.298
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307

0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.307
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.355
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.356
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362

0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.362
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.365
0.369
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371

0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.371
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377

0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.377
0.389
0.389
0.389
0.389
0.389
0.389
0.389
0.389
0.389
0.389
0.389
0.389
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393

0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.393
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.399
0.401
0.401

0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.401
0.435
0.435
0.435
0.435
0.435
0.435
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441

0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.441
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461

0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.461
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545

```
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.545
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.548
0.549
0.549
0.549
0.561
0.561
0.561
0.561
0.561
0.561
0.561
0.561
0.561
0.561
0.561
0.561
0.561
0.561
```

In [ ]: