这个单元格给助教，请忽略!

Score:

Comment:

请实现每个 function 内容，确保最终提交的notebook是可以运行的。

每一题除了必须要报告的 输出/图表，可以添加解释（中文即可）。此外可以自定义其他 function / 变量，自由添加单元格，但请确保题目中给出的 function （如第一题的 Print\_values）可以正常调用。

**Collaboration:**

TA-jiaming Zhang explained to me what is asked in problem set 2

Zhaohan Li (my roommate) explained to me what is asked in problem set 5

map和split函数学习[https://blog.csdn.net/fhy567888/article/details/136822735?](https://blog.csdn.net/fhy567888/article/details/136822735?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522B5A17EDF-B048-466C-B019-DB97F512F6BC%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=B5A17EDF-B048-466C-B019-DB97F512F6BC&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-136822735-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=map%E5%92%8Csplit&spm=1018.2226.3001.4187) [ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522B5A17EDF-B048-466C-B019-](https://blog.csdn.net/fhy567888/article/details/136822735?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522B5A17EDF-B048-466C-B019-DB97F512F6BC%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=B5A17EDF-B048-466C-B019-DB97F512F6BC&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-136822735-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=map%E5%92%8Csplit&spm=1018.2226.3001.4187) [DB97F512F6BC%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=B5A17EDF-B048-466C-B019-DB97F512F6BC&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduend~default-1-136822735-null-](https://blog.csdn.net/fhy567888/article/details/136822735?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522B5A17EDF-B048-466C-B019-DB97F512F6BC%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=B5A17EDF-B048-466C-B019-DB97F512F6BC&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-136822735-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=map%E5%92%8Csplit&spm=1018.2226.3001.4187) [null.142%5Ev100%5Epc\_search\_result\_base8&utm\_term=map%E5%92%8Csplit&spm=1018.2226.3001.4187](https://blog.csdn.net/fhy567888/article/details/136822735?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522B5A17EDF-B048-466C-B019-DB97F512F6BC%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=B5A17EDF-B048-466C-B019-DB97F512F6BC&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-136822735-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=map%E5%92%8Csplit&spm=1018.2226.3001.4187)

zip函数学习[https://blog.csdn.net/Seven\_Cloud/article/details/133359544?](https://blog.csdn.net/Seven_Cloud/article/details/133359544?ops_request_misc=%257B%2522request%255Fid%2522%253A%25225173F2C3-0BF7-4CA7-87AE-74054FB32B5F%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=5173F2C3-0BF7-4CA7-87AE-74054FB32B5F&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-133359544-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=zip%E5%87%BD%E6%95%B0python%E4%BD%9C%E7%94%A8&spm=1018.2226.3001.4187) [ops\_request\_misc=%257B%2522request%255Fid%2522%253A%25225173F2C3-0BF7-4CA7-87AE-](https://blog.csdn.net/Seven_Cloud/article/details/133359544?ops_request_misc=%257B%2522request%255Fid%2522%253A%25225173F2C3-0BF7-4CA7-87AE-74054FB32B5F%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=5173F2C3-0BF7-4CA7-87AE-74054FB32B5F&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-133359544-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=zip%E5%87%BD%E6%95%B0python%E4%BD%9C%E7%94%A8&spm=1018.2226.3001.4187)   
[74054FB32B5F%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=5173F2C3-0BF7-4CA7-87AE-](https://blog.csdn.net/Seven_Cloud/article/details/133359544?ops_request_misc=%257B%2522request%255Fid%2522%253A%25225173F2C3-0BF7-4CA7-87AE-74054FB32B5F%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=5173F2C3-0BF7-4CA7-87AE-74054FB32B5F&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-133359544-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=zip%E5%87%BD%E6%95%B0python%E4%BD%9C%E7%94%A8&spm=1018.2226.3001.4187)   
[74054FB32B5F&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~top\_positive~default-2-133359544-null-](https://blog.csdn.net/Seven_Cloud/article/details/133359544?ops_request_misc=%257B%2522request%255Fid%2522%253A%25225173F2C3-0BF7-4CA7-87AE-74054FB32B5F%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=5173F2C3-0BF7-4CA7-87AE-74054FB32B5F&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-133359544-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=zip%E5%87%BD%E6%95%B0python%E4%BD%9C%E7%94%A8&spm=1018.2226.3001.4187) [null.142%5Ev100%5Epc\_search\_result\_base8&utm\_term=zip%E5%87%BD%E6%95%B0python%E4%BD%9C%E7%94%A8&spm=1018.2226.3001.4187](https://blog.csdn.net/Seven_Cloud/article/details/133359544?ops_request_misc=%257B%2522request%255Fid%2522%253A%25225173F2C3-0BF7-4CA7-87AE-74054FB32B5F%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=5173F2C3-0BF7-4CA7-87AE-74054FB32B5F&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-2-133359544-null-null.142%5Ev100%5Epc_search_result_base8&utm_term=zip%E5%87%BD%E6%95%B0python%E4%BD%9C%E7%94%A8&spm=1018.2226.3001.4187)

1. Flowchart

Write a function Print\_values with arguments a, b, and c to reflect the following flowchart. Here the purple parallelogram operator on a list [x, y, z] is to compute and print x+y-10z. Try your output with some random a, b, and c values. Report your output when a = 10, b = 5, c = 1.

*# 1, Flowchart*   
*#*输入数据

In [

]:

value\_a**=** float(input("Please input a value\_a: ")) value\_b**=** float(input("Please input a value\_b: ")) value\_c**=** float(input("Please input a value\_c: "))

*#*检查输入的值

print("your enter value is a="**+**str(value\_a)**+**",b="**+**str(value\_b)**+**",c="**+**str(value\_c)) **if** value\_a **>** value\_b : *#*逻辑判断

**if** value\_b **>** value\_c:   
print(value\_a**+**value\_b**-**10**\***value\_c)

**else**: **if**

value\_a **>** value\_c:

print(value\_a**+**value\_c**-**10**\***value\_b **else**:

print(value\_c**+**value\_a**-**10**\***value\_b

**else**:

**if** value\_b **>** value\_c: **if** value\_a **>** value\_c:

print(value\_b**+**value\_a**-**10**\***value\_c **else**:

print(value\_b**+**value\_c**-**10**\***value\_a

**else**:

print(value\_c**+**value\_b**-**10**\***value\_a)

)

)

)

)

Report your output when a = 10, b = 5, c = 1: 5

输入数据，然后先比较a,b。然后比较c,d。输入a = 10, b = 5, c = 1，输出结果为5。

2. Continuous ceiling function

Given a list with N positive integers. For every element F(ceil(x/3)) + 2x, where F(1) = 1.

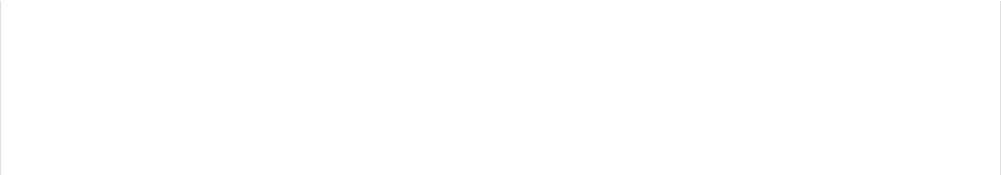
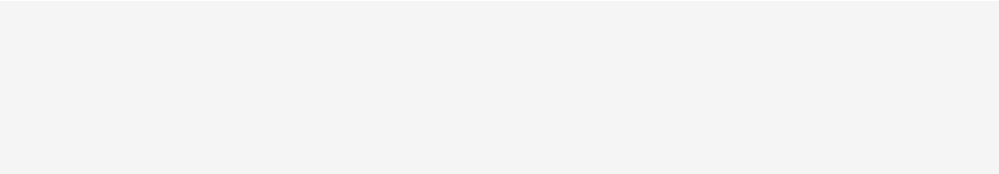
x of the list, find the value of continuous ceiling function defined as

F(x) =

*#2. Continuous ceiling function*   
*#*定义函数

In [4]:

**import** numpy **as** np

**def** F(x): **if** x**==**1:

*TA-jiaming Zhang explained to me what is asked in problem set 2*

**return** 1 **else**:

**return** F(np**.**ceil(x**/**3))**+**2**\***x

*#*输入数据

user\_input**=** input("enter a number list(数字由英文逗号隔开）")

*#*转换成列表

numbers **=** list(map(int, user\_input**.**split(',')))

*#* 计算每个元素 *x* 对应的 *F(x)* 并存储结果 results **=** [F(x) **for** x **in** numbers]

*#* 输出结果

**for** x, result **in** zip(numbers, results): print(f"F({x}) = {result}")

F(32) = 99.0 F(46) = 141.0 F(54) = 161.0

First, give a function F(x) and use the if loop. Then," numbers = list(map(int, user\_input.split(',')))" uses the map and split function to transfer user input to the list so that we can run F(x) one by one. when input (32,46,54) ,result is F(32)=99,F(46)=141,F(54)=161.

1. Dice rolling

**3.1** Given 10 dice each with 6 faces, numbered from 1 to 6. Write a function Find\_number\_of\_ways to find the number of ways to get sum x, defined as the sum of values on each face when all the dice are thrown.

*# 3.1 Dice rolling*

*#* 定义函数找到和为*x*的方式数量

**def** Find\_number\_of\_ways(dice, target\_sum): **if** dice **==** 0:

**if** target\_sum **==** 0: *#*如果和为*0*，筛子数为*0*，则返回*1*，有一种情况 **return** 1

**else**: *#*如果和不为*0*，筛子数为*0*，则返回*0*，没有这种情况

0

**if** target\_sum **<** dice **or** target\_sum **>** dice **\*** 6: *#*如果和小于筛子数或者和大于筛子数与最大面数乘积，返回*0*，也不存在这种情况

**return** 0

total\_ways **=** 0

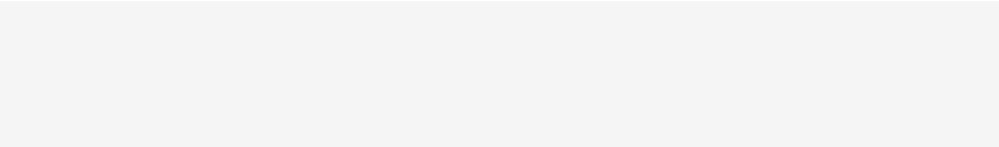
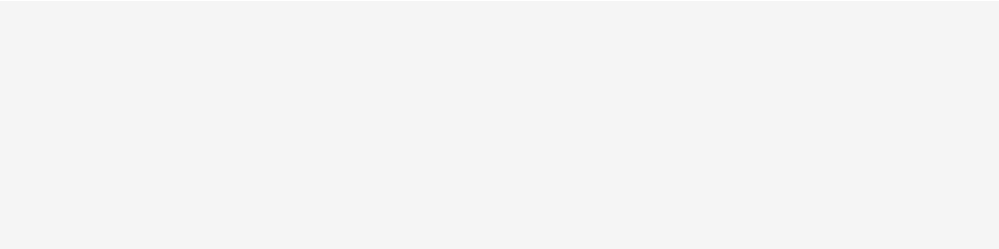
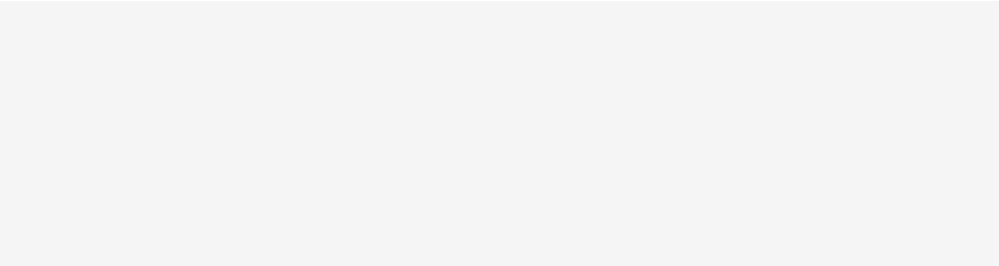
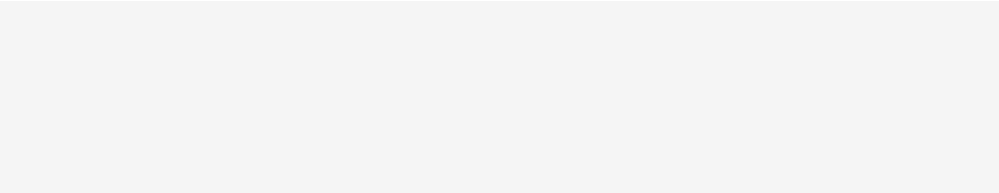
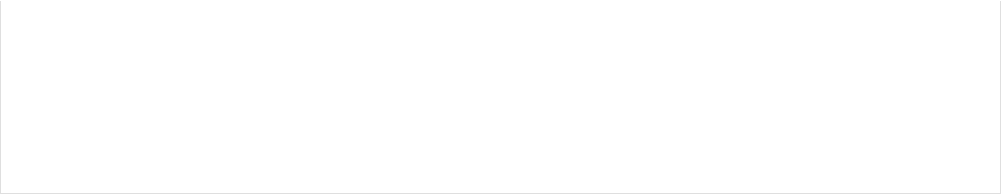
**for** i **in** range(1, 7):

total\_ways **+=** Find\_number\_of\_ways(dice **-** 1, target\_sum **-** i) *#*递归函数，*Find\_number\_of\_ways*（*0*，*target\_sum)*是递归终止。

**return** total\_ways

[1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228, 147940, 243925, 383470, 576565, 831204, 1151370, 1535040, 1972630, 2446300, 29

30455, 3393610, 3801535, 4121260, 4325310, 4395456, 4325310, 4121260, 3801535, 3393610, 2930455, 2446300, 1972630, 1535040, 1151370, 831204,   
576565, 383470, 243925, 147940, 85228, 46420, 23760, 11340, 4995, 2002, 715, 220, 55, 10, 1]

最大的方式数量是 4395456，对应的和是 35

In [5]:

In [ ]:

*# +10* 是因为 *x* 从 *10* 开始计算

In [ ]:

**for**   **in** range(N)] *#*下划线代表是个虚值，只需要重复*N*次就可以了

*0* 到 *10*

**3.2** Count the number of ways for any x from 10 to 60, assign the number of ways to a list called Number\_of\_ways, so which x yields the maximum of Number\_of\_ways?

*# 3.2*

*#* 计算从*10*到*60*的方式数量 dice **=** 10 *#* 骰子的数量 Number\_of\_ways **=** []

**for** x **in** range(10, 61): *#* 遍历 *x* 从 *10* 到 *60* ways **=** Find\_number\_of\_ways(dice, x) Number\_of\_ways**.**append(ways)

print(Number\_of\_ways)

*#* 找到最大方式数量以及对应的*x*

max\_ways **=** max(Number\_of\_ways) *#*找最大值

x\_with\_max\_ways **=** 10 **+** Number\_of\_ways**.**index(max\_ways)

print(f"最大的方式数量是 {max\_ways}，对应的和是 {x\_with\_max\_ways}")

运行结果为最大的方式数量是 4395456，对应的和是 35

解释：3.1：定义函数，采用迭代计算的方法，依次从1到6中减少一个数字并且在target\_sum中减小对应的值。然后进行第二次迭代，第三次迭代···直到当dice （骰子数）为0，判断target\_sum是都恰好为0，若是，则发现一种方法；若不是，则返回0，这条路走不通。如果target\_sum小于骰子数或者大于6乘以骰子   
数（即最大值）都不行，返回0.

3.2：对10到60的target\_sum进行遍历，带入方程中寻找对应的路径数，然后用max函数找到这些路径数中的最大值与对应的x。

1. Dynamic programming

**4.1 [5 points]** Write a function Random\_integer to fill an array of N elements by randomly selecting integers from 0 to 10.

*#4. Dynamic programming*（第三题代码为完整代码，方便老师或*TA*运行）

**import** random

**import** matplotlib.pyplot **as** plt

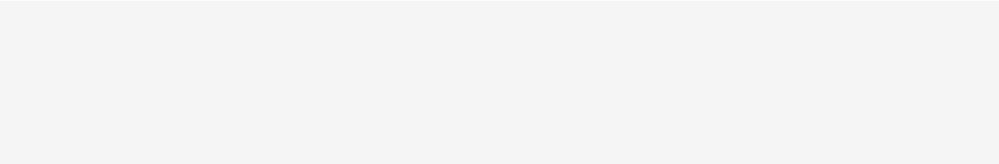
*#* 生成包含 *N* 个随机整数的数组，范围从 **def** Random\_integer(N):

**return** [random**.**randint(0, 10)

**4.2 [15 points]** Write a function Sum\_averages to compute the sum of the average of all subsets of the array. For example, given an array of [1,   
2, 3], you Sum\_averages function should compute the sum of: average of [1], average of [2], average of [3], average of [1, 2],   
average of [1, 3], average of [2, 3], and average of [1, 2, 3].

In [

]:



*#* 优化的 *Sum\_averages*，直接计算所有子集平均值和 **def** Sum\_averages(array):

N **=** len(array)

**if** N **==** 0: **return** 0

*#* 计算数组元素的总和

total\_sum **=** sum(array)

*#* 计算所有子集的平均值和

**return** (total\_sum **\*** (2**\*\***(N**-**1))) **/** N *#*运算子集平均值的和的公式

**4.3 [5 points]** Call Sum\_averages with N increasing from Total\_sum\_averages, describe what you see.

1 to

100, assign the output to a list called

Total\_sum\_averages. Plot

In [17]:

*0* 到 *10*

**for**   **in** range(N)] *#*下划线代表是个虚值，只需要重复*N*次就可以了

*#* 优化的 *Sum\_averages*，直接计算所有子集平均值和 **def** Sum\_averages(array):

N **=** len(array)

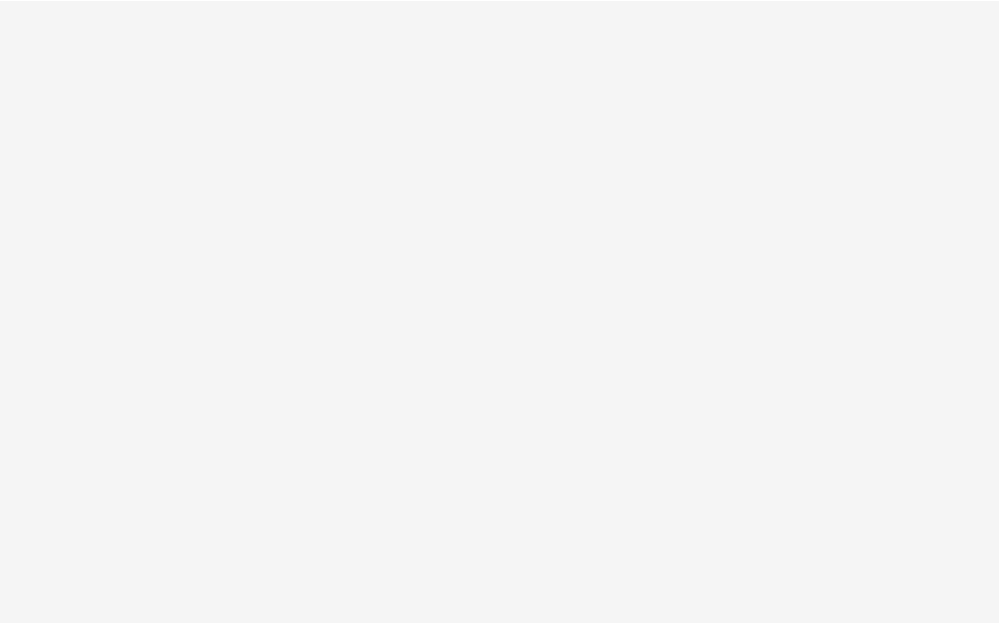
**if** N **==** 0: **return** 0

*#* 计算数组元素的总和

total\_sum **=** sum(array)

*#* 计算所有子集的平均值和

**return** (total\_sum **\*** (2**\*\***(N**-**1))) **/** N *#*运算子集平均值的和的公式



*#4.3 Dynamic programming*

**import** random

**import** matplotlib.pyplot **as** plt

*#* 生成包含 *N* 个随机整数的数组，范围从 **def** Random\_integer(N):

**return** [random**.**randint(0, 10)

marker**=**'o')

*#* 存储每个 *N* 对应的 *Sum\_averages* 结果

Total\_sum\_averages **=** []

*#* 调用 *Sum\_averages*，*N* 从 *1* 到 *100* **for** N **in** range(1, 101):

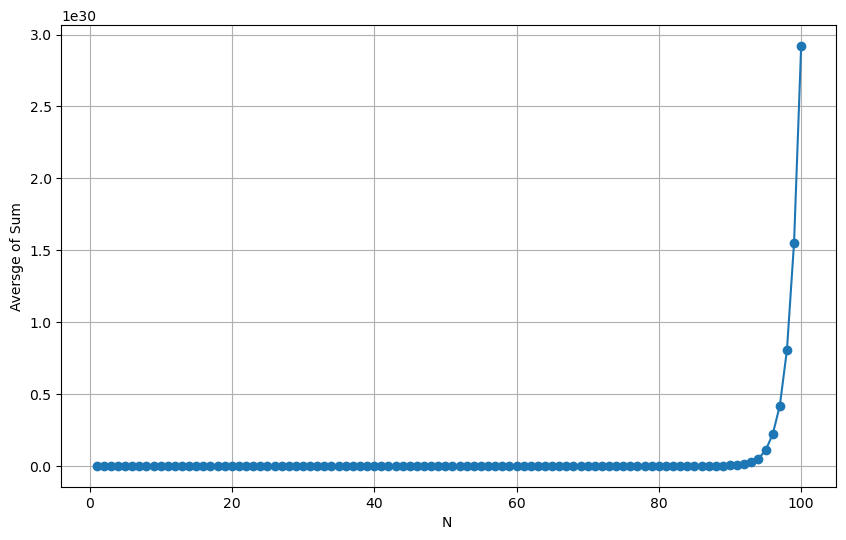
random\_array **=** Random\_integer(N) total\_sum **=** Sum\_averages(random\_array) Total\_sum\_averages**.**append(total\_sum)

*#* 绘制结果

plt**.**figure(figsize**=**(10, 6))   
plt**.**plot(range(1, 101), Total\_sum\_averages, plt**.**xlabel('N ')

plt**.**ylabel('Aversge of Sum')

plt**.**grid() plt**.**show()



平均值在0-80都比较稳定，接近0，但是从85开始陡峭上升，100时停留在3.00左右。这与公式中的2\*\*（N-1)有关。

1. Path counting

In [

]:

**5.1 [5 points]** Create a matrix with N rows and M columns, fill the right-bottom corner and top-left corner cells with rest of matrix with integer 0 or 1.

1 , and randomly fill the

*#5. Path counting*

**import** random   
**def** create\_matrix(N,

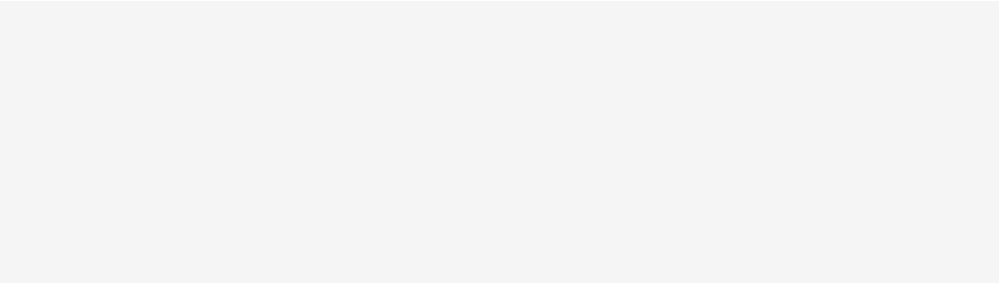
*Zhaohan Li (my roommate) explained to me what is asked in problem set 5*（第三题代码为完整代码，方便老师或*TA*运行）

M):

matrix **=** [[0 **for**   **in** range(M)]**for**   **in** range(N)] *#*建立一个空的二维矩阵

**for** i **in** range(N): **for** j **in** range(M):

matrix[i][j]**=**random**.**randint(0,1) *#*在每行进行*0*，*1*的随机填充



matrix[0][0] **=** 1

matrix[N**-**1][M**-**1] **=** 1

**return** matrix   
print(" 实验(10，8)的结果

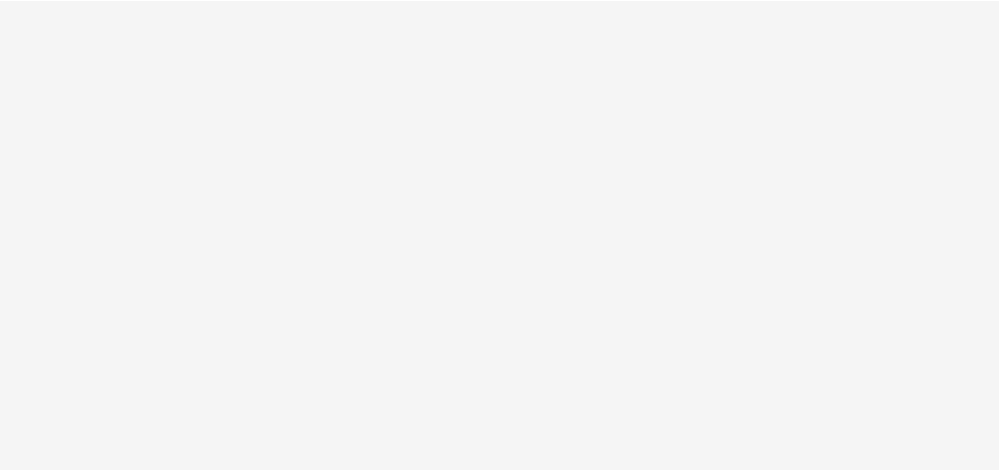
*#* 设置左上角和右下角为 *1*

" **+** str(create\_matrix(10,8)) ) *#*试验

**5.2 [25 points]** Consider a cell marked with 0 as a blockage or dead-end, and a cell marked with 1 is good to go. Write a function Count\_path to count the total number of paths to reach the right-bottom corner cell from the top-left corner cell.

**Notice:** for a given cell, you are **only allowed** to move either rightward or downward.

In [ ]:



**def** Count\_path(matrix): *#*定义函数   
N **=** len(matrix)   
M **=** len(matrix[0])   
*#* 创建一个与矩阵大小相同的路径计数矩阵，下划线表示虚值 dp **=** [[0] **\*** M **for**   **in** range(N)]   
   
*#* 初始化左上角   
dp[0][0] **=** 1

*#* 填充第一列的路径数 **for** i **in** range(1, N):

**if** matrix[i][0] **==** 1: dp[i][0] **=** dp[i**-**1][0]

*#* 填充第一行的路径数 **for** j **in** range(1, M):

**if** matrix[0][j] **==** 1: dp[0][j] **=** dp[0][j**-**1]

*#* 填充其余位置的路径数 **for** i **in** range(1, N):

**for** j **in** range(1, M): **if** matrix[i][j] **==** 1:

dp[i][j] **=** dp[i**-**1][j] **+** dp[i][j**-**1]

*#* 右下角的值就是到达右下角的所有路径数 **return** dp[N**-**1][M**-**1]

**5.3 [5 points]** Let N = 10, M = 8, run Count\_path for 1000 times, each time the matrix (except the right-bottom corner and top-left corner   
cells, which remain being 1) is re-filled with integer 0 or 1 randomly, report the mean of total number of paths from the 1000 runs.

In [3]:

*Zhaohan Li (my roommate) explained to me what is asked in problem set 5*（第三题代码为完整代码，方便老师或*TA*运行）

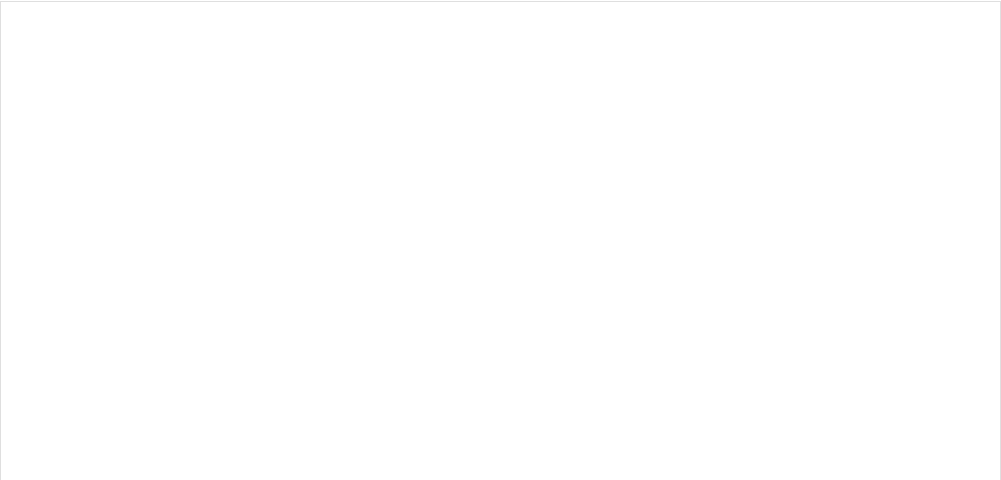
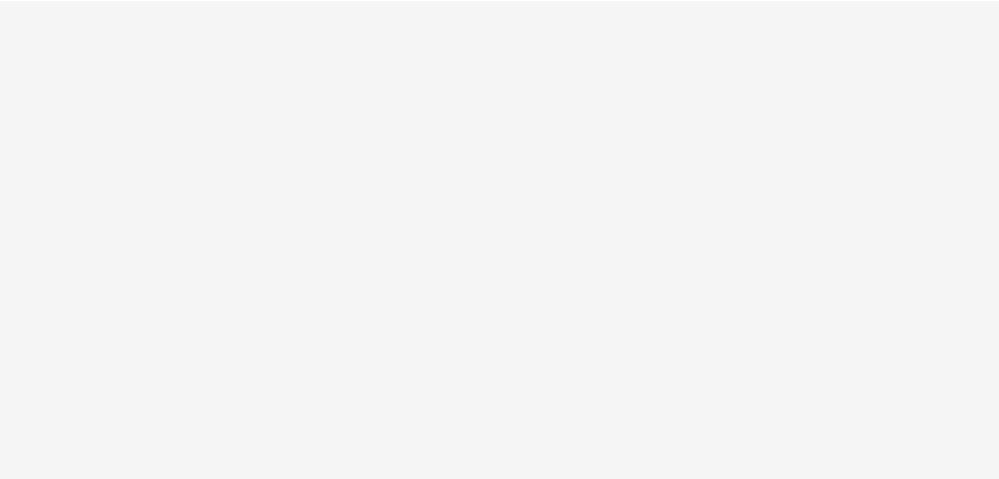
matrix **=** [[0 **for**   **in** range(M)]**for**   **in** range(N)] *#*建立一个空的二维矩阵

**for** i **in** range(N): **for** j **in** range(M):

matrix[i][j]**=**random**.**randint(0,1) *#*在每行进行*0*，*1*的随机填充

matrix[0][0] **=** 1

matrix[N**-**1][M**-**1] **=** 1

**return** matrix

*#5. Path counting*

**import** random   
**def** create\_matrix(N, M):

*#* 设置左上角和右下角为

*1*

print(" 实验(10，8)的结果

" **+** str(create\_matrix(10,8)) ) *#*试验

**def** Count\_path(matrix): *#*定义函数 N **=** len(matrix)

M **=** len(matrix[0])

*#* 创建一个与矩阵大小相同的路径计数矩阵，下划线表示虚值 dp **=** [[0] **\*** M **for**   **in** range(N)]

*#* 初始化左上角 dp[0][0] **=** 1

*#* 填充第一列的路径数 **for** i **in** range(1, N):

**if** matrix[i][0] **==** 1: dp[i][0] **=** dp[i**-**1][0]

*#* 填充第一行的路径数 **for** j **in** range(1, M):

**if** matrix[0][j] **==** 1: dp[0][j] **=** dp[0][j**-**1]

*#* 填充其余位置的路径数

**for** i **in** range(1, N): **for** j **in** range(1, M):

**if** matrix[i][j] **==** 1:   
dp[i][j] **=** dp[i**-**1][j] **+**

*#* 右下角的值就是到达右下角的所有路径数 **return** dp[N**-**1][M**-**1]

*#5.3*

average\_paths **=** 0

**for**   **in** range(1001): *#*运行一千次 matrix**=**create\_matrix(10,8) paths **=** Count\_path(matrix)

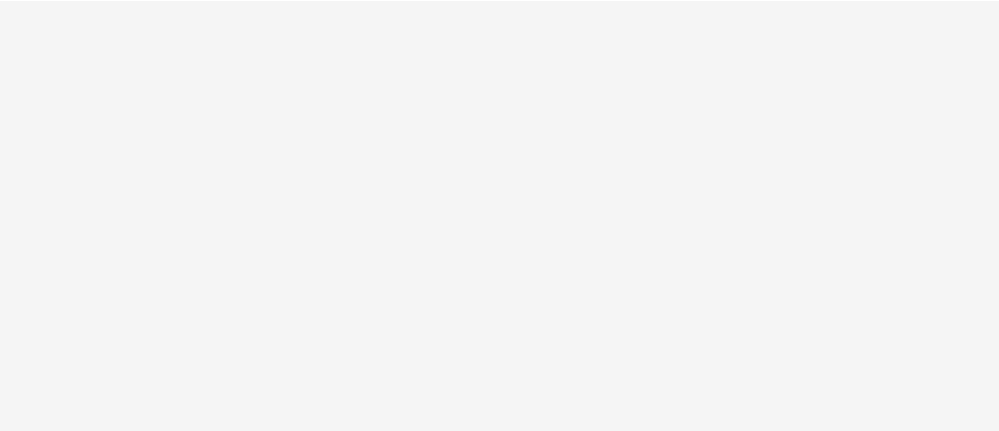
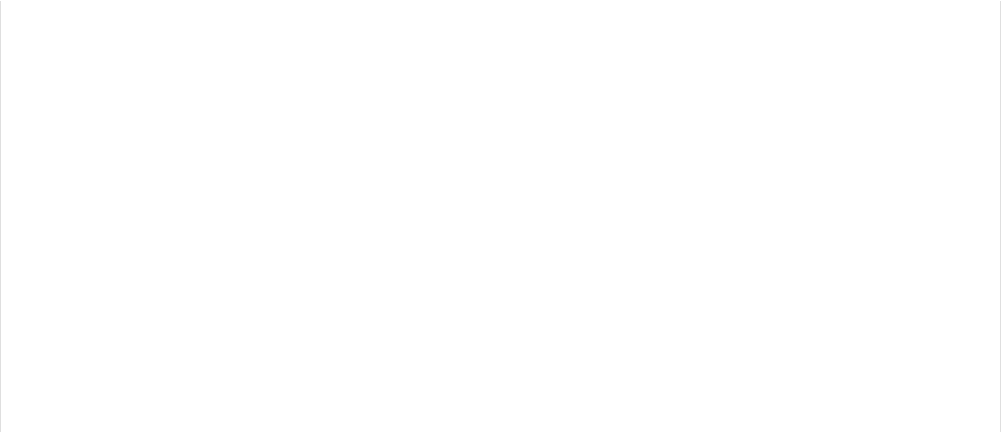
average\_paths **+=**paths *#*计算总和

print("your paths 1000 times , get the answer " ,average\_paths**/**1000)

实验(10，8)的结果 [[1, 1, 0, 1, 0, 0, 0, 1], [1, 1, 0, 1, 0, 0, 1, 1], [1, 1, 1, 1, 0, 1, 0, 1], [0, 0, 1, 1, 0, 0, 1, 0], [1, 1, 0, 1, 1,   
0, 1, 0], [1, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 1, 1], [0, 1, 0, 0, 0, 1, 1, 1], [1, 1, 1, 1, 0, 0, 1, 1], [0, 1, 0, 0, 0, 0, 0, 1]] your paths 1000 times , get the answer 0.338

实验(10，8)的结果 [[1, 1, 0, 1, 0, 0, 0, 1], [1, 1, 0, 1, 0, 0, 1, 1], [1, 1, 1, 1, 0, 1, 0, 1], [0, 0, 1, 1, 0, 0, 1, 0], [1, 1, 0, 1, 1, 0, 1, 0], [1, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 1, 0, 1, 1], [0, 1, 0, 0, 0, 1, 1, 1], [1, 1, 1, 1, 0, 0, 1, 1], [0, 1, 0, 0, 0, 0, 0, 1]] your paths 1000 times , get the answer 0.338

解释：5.1，先创造一个空的二位矩阵，然后用一个两层for循环来填充随机数。最后把左上和右下改成1. 5.2，创建一个与矩阵大小相同的路径计数矩阵，依次 填充第一列和第一行的路经数。最后改变其他位置的路径书， 5.3，运行1000次后，获得的平均值为0.338，每次运行的结果不一样，因为随机数列的存在。



dp[i][j**-**1]