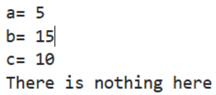
Assignment 1

谢婷 12532753

Problem 1：input the values for a, b, and c. Then, based on the flowchart, determine which elements in the [x, y, z] list correspond to a, b, and c respectively. Finally, calculate x + y - 10z.

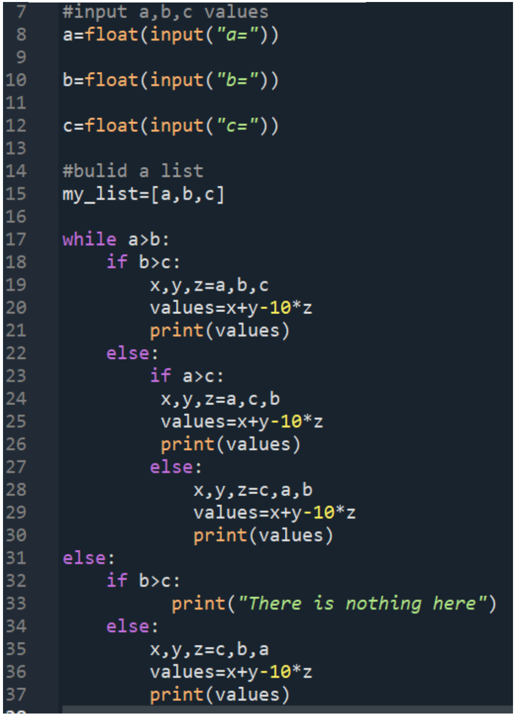
For example,input a=5,b=15,c=10.a>b is False,then b>c is True.Finally output There is nothing here

Output:



```
a= 5
b= 15
c= 10
There is nothing here
```

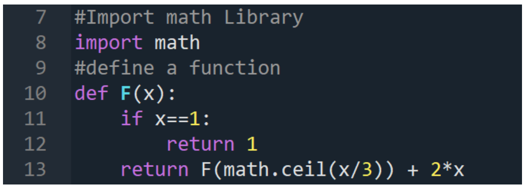
Code:



```python
#input a,b,c values
a=float(input("a="))

b=float(input("b="))

c=float(input("c="))

#bulid a list
my_list=[a,b,c]

while a>b:
    if b>c:
        x,y,z=a,b,c
        values=x+y-10*z
        print(values)
    else:
        if a>c:
         x,y,z=a,c,b
         values=x+y-10*z
         print(values)
        else:
            x,y,z=c,a,b
            values=x+y-10*z
            print(values)
else:
    if b>c:
          print("There is nothing here")
    else:
        x,y,z=c,b,a
        values=x+y-10*z
        print(values)
```

Problem 2：First import the math library. Define F(x) = F(ceil(x/3)) + 2x, where F(1) = 1.

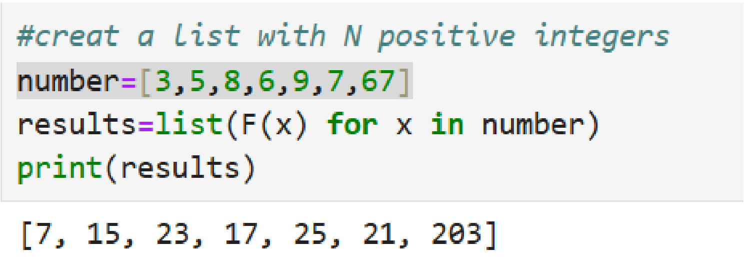
As follows:



```python
#Import math Library
import math
#define a function
def F(x):
    if x==1:
        return 1
    return F(math.ceil(x/3)) + 2*x
```

To demonstrate the code, consider a list of positive integers: number=[3,5,8,6,9,7,67] The result is[7,15,23,17,25,21,203]

As follows:



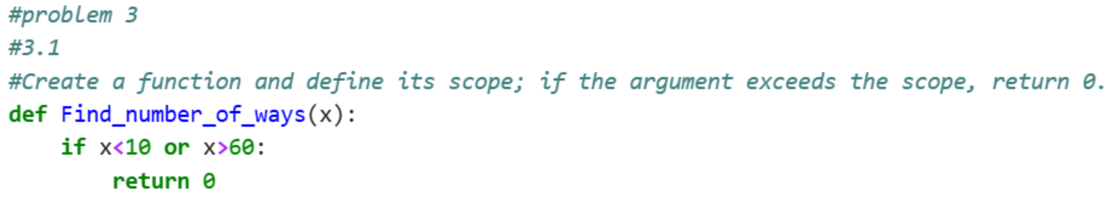
```
#creat a list with N positive integers
number=[3,5,8,6,9,7,67]
results=list(F(x) for x in number)
print(results)
```

```
[7, 15, 23, 17, 25, 21, 203]
```

Problem 3：

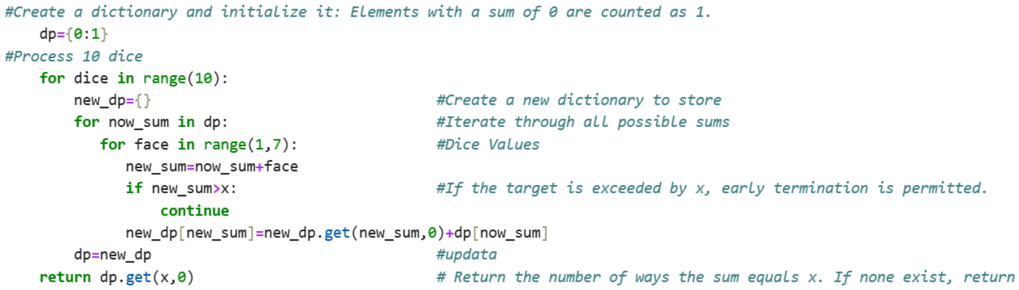
Draw lessons from: #https://blog.csdn.net/cumtb2002/article/details/107764190

3.1 First, we need to define the range of values for x in the function. When summing the values of 10 dice, 10 < x < 60.



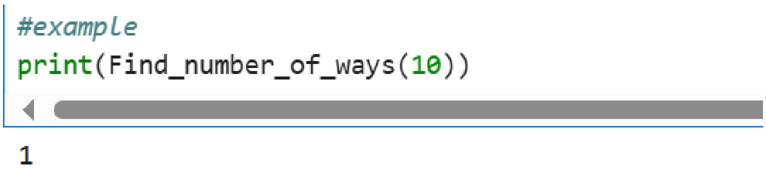
```
#problem 3
#3.1
#Create a function and define its scope; if the argument exceeds the scope, return 0.
def Find_number_of_ways(x):
    if x<10 or x>60:
        return 0
```

Initial state design: The number of combinations with a sum of 0 is 1 (i.e., no dice are rolled). Process 10 dice one by one, creating a new dictionary with `new_dp` to store results. Use two for loops and one if statement for optimization to reduce computation. If the new sum exceeds the target `x`, skip it. Then proceed with assignment. Finally, `dp.get(x, 0)` returns the number of combinations corresponding to x. If none exist, it returns 0.



```
#Create a dictionary and initialize it: Elements with a sum of 0 are counted as 1.
    dp={0:1}
#Process 10 dice
    for dice in range(10):
        new_dp={}                                    #Create a new dictionary to store
        for now_sum in dp:                           #Iterate through all possible sums
           for face in range(1,7):                   #Dice Values
              new_sum=now_sum+face
              if new_sum>x:                          #If the target is exceeded by x, early termination is permitted.
                  continue
              new_dp[new_sum]=new_dp.get(new_sum,0)+dp[now_sum]
        dp=new_dp                                    #updata
    return dp.get(x,0)                               # Return the number of ways the sum equals x. If none exist, return
```

For example, when x=10, there is only one possible outcome: all 10 dice showing a 1. The code output is also 1.



```
#example
print(Find_number_of_ways(10))
```

```
1
```

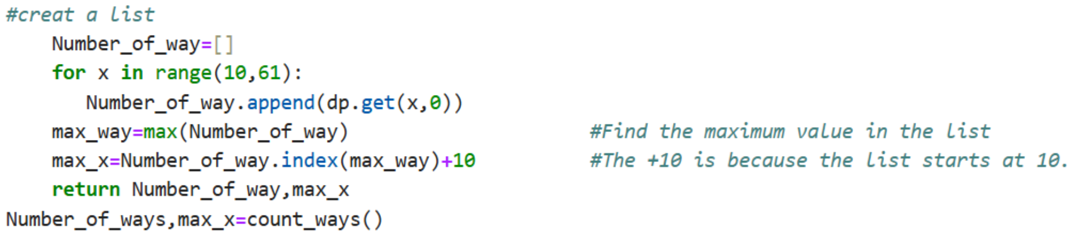
3.2 Building upon Section 3.1, initialize an empty list named \`Number_of_way\` . Iterate through x from 10 to 60, adding \`dp.get(x, 0)\` (i.e., the number of combinations for x) to the list. Ultimately, \`Number_of_way\` becomes a list of length 51 (61 - 10 = 51):

Number_of_way[0] corresponds to the number of combinations for x=10.

Number_of_way[1] corresponds to the number of combinations for x=11

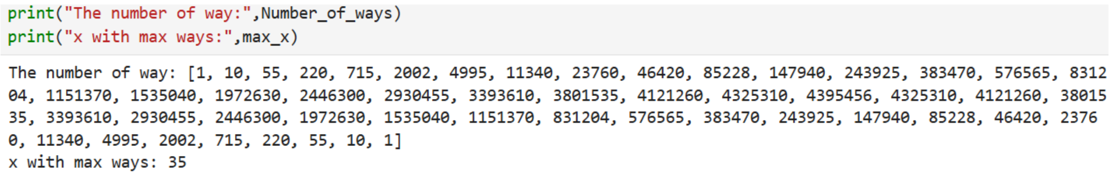
....

Number_of_way[50] corresponds to the number of combinations for x=60.



```
#creat a list
    Number_of_way=[]
    for x in range(10,61):
       Number_of_way.append(dp.get(x,0))
    max_way=max(Number_of_way)                       #Find the maximum value in the list
    max_x=Number_of_way.index(max_way)+10            #The +10 is because the list starts at 10.
    return Number_of_way,max_x
Number_of_ways,max_x=count_ways()
```

Finally, output is as follows:



```
print("The number of way:",Number_of_ways)
print("x with max ways:",max_x)
```

```
The number of way: [1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228, 147940, 243925, 383470, 576565, 8312
04, 1151370, 1535040, 1972630, 2446300, 2930455, 3393610, 3801535, 4121260, 4325310, 4395456, 4325310, 4121260, 38015
35, 3393610, 2930455, 2446300, 1972630, 1535040, 1151370, 831204, 576565, 383470, 243925, 147940, 85228, 46420, 2376
0, 11340, 4995, 2002, 715, 220, 55, 10, 1]
x with max ways: 35
```

When the sum is 35, the number of ways obtained is the greatest.(4395456).

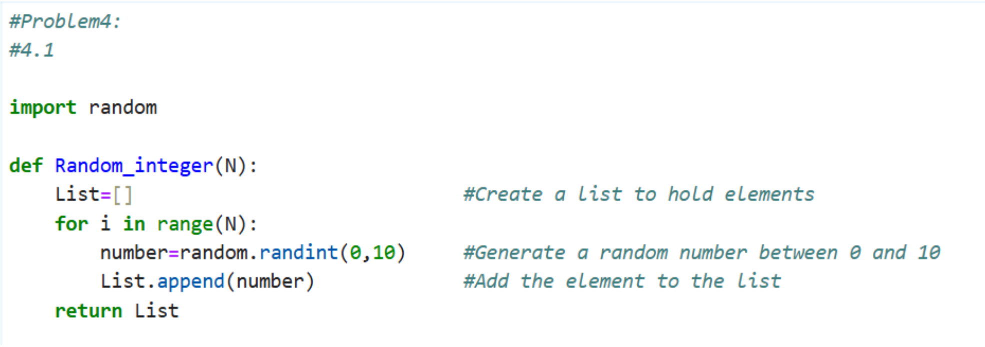
Problem 4: 4.1

This code defines a function called Random_integer that generates a list of N random integers, each ranging from 0 to 10 (inclusive). Here's how it works:

1. Initialize an empty list List to store the generated random numbers.

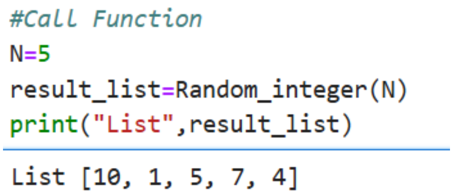
2. Loop N times, each time calling random.randint(0, 10) to generate a random integer between 0 and 10, and append it to List.

3. Return the generated list List.



```
#Problem4:
#4.1

import random

def Random_integer(N):
    List=[]                       #Create a list to hold elements
    for i in range(N):
        number=random.randint(0,10)      #Generate a random number between 0 and 10
        List.append(number)              #Add the element to the list
    return List
```

For example

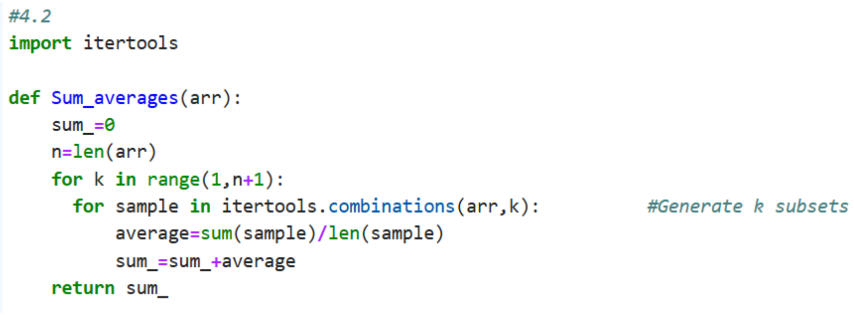


```
#Call Function
N=5
result_list=Random_integer(N)
print("List",result_list)
```

```
List [10, 1, 5, 7, 4]
```

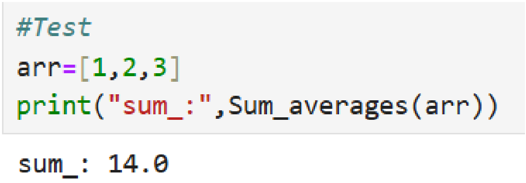
4.2

A function named `Sum_averages` is defined to compute the sum of the averages of all non-empty subsets in an array. Specifically, given an array, this function will: iterate over all possible subset sizes (from 1 to the array length `n`). Generate all subsets of that size (using `itertools.combinations`). Calculate the average of each subset (sum of subset elements divided by the subset size). Accumulate the averages of all subsets and ultimately return the total sum.



```
#4.2
import itertools

def Sum_averages(arr):
    sum_=0
    n=len(arr)
    for k in range(1,n+1):
      for sample in itertools.combinations(arr,k):          #Generate k subsets
          average=sum(sample)/len(sample)
          sum_=sum_+average
    return sum_
```

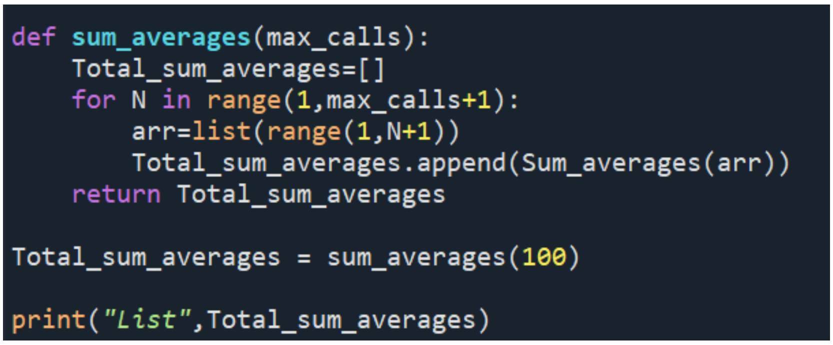
For example: When the list is [1,2,3], the output result is 14.



```
#Test
arr=[1,2,3]
print("sum_:",Sum_averages(arr))
```

```
sum_: 14.0
```

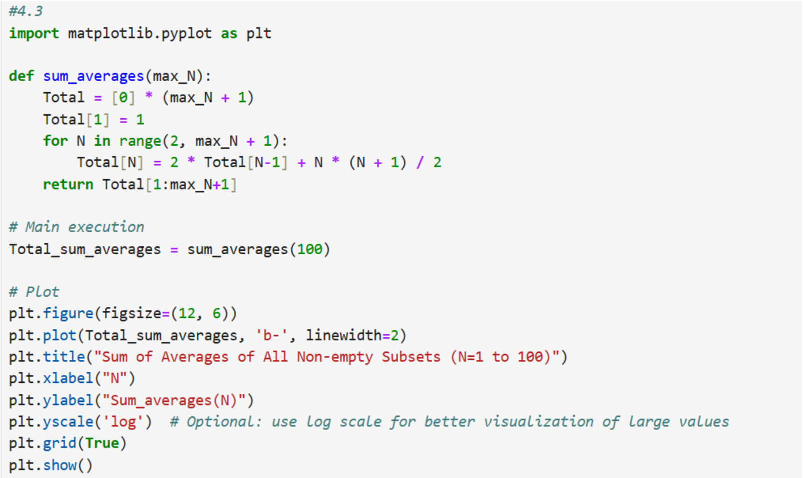
4.3 Initially, we employed brute force, but found the program's runtime was excessively long, only suitable for N < 20. Since N = (1, 100) in this problem, it is not applicable here. The program is as follows:



```
def sum_averages(max_calls):
    Total_sum_averages=[]
    for N in range(1,max_calls+1):
        arr=list(range(1,N+1))
        Total_sum_averages.append(Sum_averages(arr))
    return Total_sum_averages

Total_sum_averages = sum_averages(100)

print("List",Total_sum_averages)
```

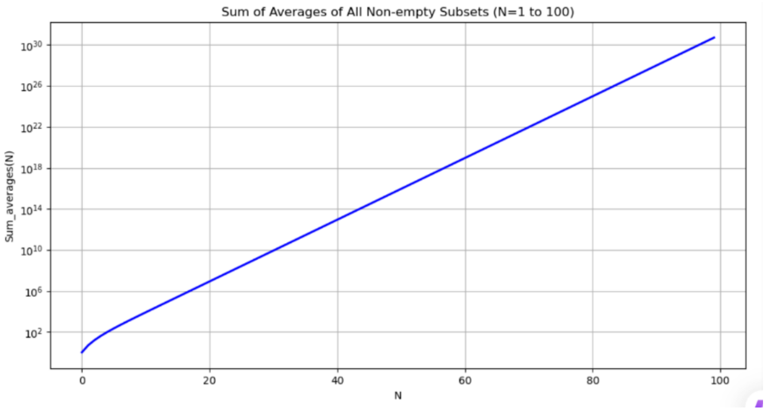
The following methods are referenced from Wenxin Yiyan:

We adopted a recursive approach based on dynamic programming. By analyzing the distribution patterns of subset averages, this method derives the recursive formula.



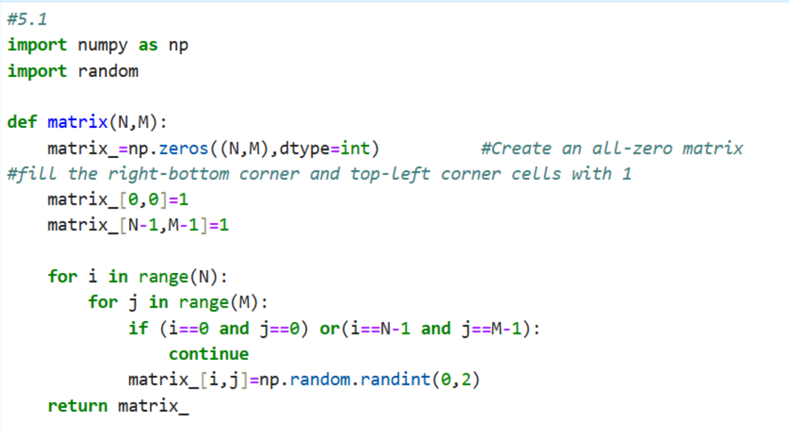
```
#4.3
import matplotlib.pyplot as plt

def sum_averages(max_N):
    Total = [0] * (max_N + 1)
    Total[1] = 1
    for N in range(2, max_N + 1):
        Total[N] = 2 * Total[N-1] + N * (N + 1) / 2
    return Total[1:max_N+1]

# Main execution
Total_sum_averages = sum_averages(100)

# Plot
plt.figure(figsize=(12, 6))
plt.plot(Total_sum_averages, 'b-', linewidth=2)
plt.title("Sum of Averages of All Non-empty Subsets (N=1 to 100)")
plt.xlabel("N")
plt.ylabel("Sum_averages(N)")
plt.yscale('log')  # Optional: use log scale for better visualization of large values
plt.grid(True)
plt.show()
```

The graph is shown below, revealing that Sum_averages(N) exhibits exponential growth.



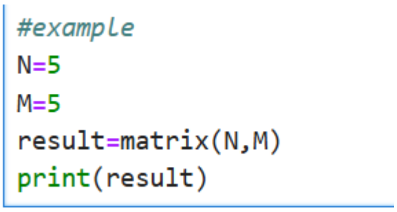
Problem5：

5.1 First, create an N-row by M-column zero matrix. Then, set the top-left and top-right corners to 1. Use a for loop to randomly fill the remaining cells with 0 and 1.

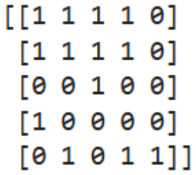


```
#5.1
import numpy as np
import random

def matrix(N,M):
    matrix_=np.zeros((N,M),dtype=int)          #Create an all-zero matrix
#fill the right-bottom corner and top-left corner cells with 1
    matrix_[0,0]=1
    matrix_[N-1,M-1]=1

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

For example:N=5,M=5



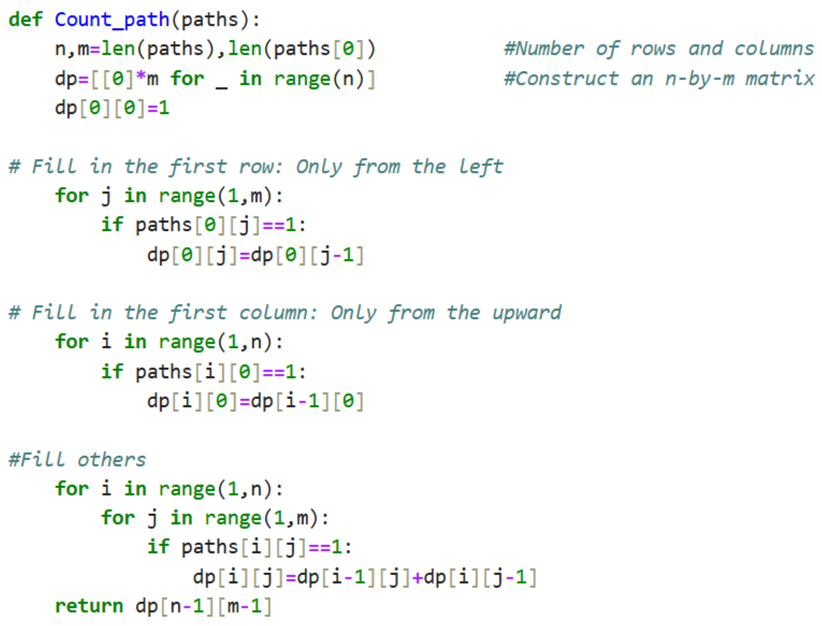
```
#example
N=5
M=5
result=matrix(N,M)
print(result)
```



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

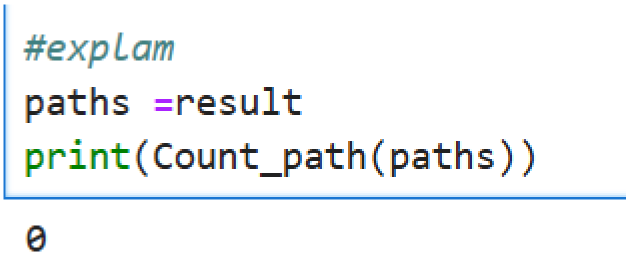
5.2:

This code uses dynamic programming to calculate the number of distinct paths from the top-left corner (0, 0) to the bottom-right corner (n-1, m-1) of a two-dimensional grid paths, where paths[i][j] = 1 indicates a passable position and 0 indicates an obstacle. The algorithm first initializes a DP table to record path counts, setting the path count from the starting point (0, 0) to 1; Next, it handles boundary conditions: the first row can only move left-to-right, and the first column can only move top-to-bottom, inheriting path counts from the left or above, respectively. For other positions, if passable, the path count is the sum of the upper and left path counts; if impassable, the path count remains 0. Finally, it returns the value of `dp[n-1][m-1]` , representing the total number of paths reaching the endpoint.



```
def Count_path(paths):
    n,m=len(paths),len(paths[0])           #Number of rows and columns
    dp=[[0]*m for _ in range(n)]           #Construct an n-by-m matrix
    dp[0][0]=1

# Fill in the first row: Only from the left
    for j in range(1,m):
        if paths[0][j]==1:
            dp[0][j]=dp[0][j-1]

# Fill in the first column: Only from the upward
    for i in range(1,n):
        if paths[i][0]==1:
            dp[i][0]=dp[i-1][0]

#Fill others
    for i in range(1,n):
        for j in range(1,m):
            if paths[i][j]==1:
                dp[i][j]=dp[i-1][j]+dp[i][j-1]
    return dp[n-1][m-1]
```

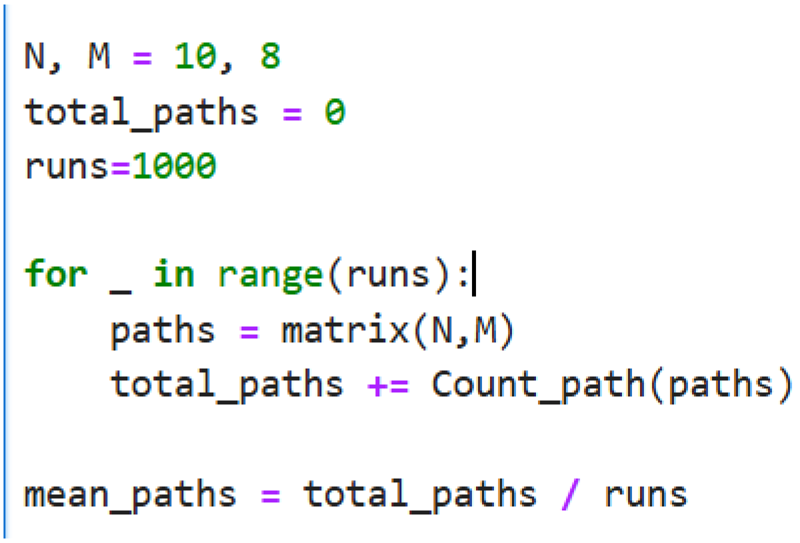
For example: Based on 5.1   N=5,M=5,



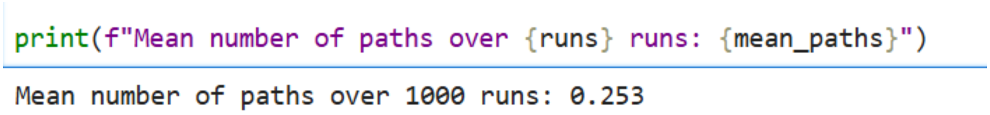
```
#explam
paths =result
print(Count_path(paths))
```

```
0
```

5.3 Based on 5.1 and 5.2,signed N=10,M=8,runs=1000,then get the answer ,such as mean number of paths over 1000 runs: 0.253.



```
N, M = 10, 8
total_paths = 0
runs=1000

for _ in range(runs):
    paths = matrix(N,M)
    total_paths += Count_path(paths)

mean_paths = total_paths / runs
```



```python
print(f"Mean number of paths over {runs} runs: {mean_paths}")
```

```
Mean number of paths over 1000 runs: 0.253
```