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:

Collaboration on solving the assignment is allowed, after you have thought about the problem sets on your own. It is also OK to get clarification (but not solutions) from online resources, again after you have thought about the problem sets on your own.

There are two requirements for collaboration:

- Cite your collaborators **fully and completely** (*e.g.*, "XXX explained to me what is asked in problem set 3"). Or cite online resources (*e.g.*, "I got inspired by reading XXX") that helped you.
- Write your scripts and report **independently** - the scripts and report must come from you only.

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`.

In [1]:
```
def Print_values(a,b,c):
    x,y,z=0,0,0
    if a>b:
        if b>c:
            x,y,z=a,b,c
        else:
            if a>c:
                x,y,z=a,c,b
            else:
                x,y,z=c,a,b
    else:
        if b>c:
            return
        else:
```

```
            x,y,z=c,b,a
    result=x+y-10*z
    return result
print("a=10,b=5,c=1时结果是",Print_values(a=10,b=5,c=1))
```

```
a=10,b=5,c=1时结果是 5
```

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

[a=10,b=5,c=1时结果是 5] 应用if语句让x,y,z取得对应的a,b,c

2. Continuous ceiling function

Given a list with `N` positive integers. For every element `x` of the list, find the value of continuous ceiling function defined as `F(x) = F(ceil(x/3)) + 2x` , where `F(1) = 1` .

In [4]:
```
import math
#定义函数
def F(x):
    if x == 1:
        return 1
    else:
        return F(math.ceil(x / 3)) + 2 * x
#给列表N和结果输出列表RESULT
N=[1,2,3,4,5,6,7,8,9]
RESULT = []
#遍历列表并输出
for i in N:
    RESULT.append(F(i))
print(RESULT)
```

```
[1, 5, 7, 13, 15, 17, 21, 23, 25]
```

[结果：[1, 5, 7, 13, 15, 17, 21, 23, 25]] 点定义函数，再设置列表，遍历列表输出答案

3. 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.

In [6]:
```
import random
#定义函数
def Find_number_of_ways(sum_x):
    x=0
    #计算总的骰子的和
    for i in range(sum_x):
        #计算每个骰子的面的和
        for j in range(6):
            x+=random.randint(1, 6)
    return x
```

```
#计算10的骰子的和
print(Find_number_of_ways(10))
```

```
217
```

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`?

In [8]:
```
#创建列表
Number_of_ways=[]
#编辑10~60个骰子的面数和的列表
for sum_x in range(10, 61):
    Number_of_ways.append(Find_number_of_ways(sum_x))
#找出最大值
maximum = max(Number_of_ways)
#找出最大值对应的骰子数
max_x=Number_of_ways.index(maximum)+10
print("x的最大值是"+str(max_x))
```

```
x的最大值是60
```

So which `x` yields the maximum of `Number_of_ways`? [60]

[用循环，先计算每个骰子的面的和，再计算总的骰子的和]

4. 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`.

In [11]:
```
import random
#定义函数
def Random_integer(N):
    #创建空列表存储结果
    result = []
    #循环N次
    for i in range(N):
        result.append(random.randint(0, 10))
    return result
#结果输出
print(Random_integer(int(input("输入一个整数N:"))))
```

```
[1, 6, 8, 2]
```

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 [23]:
```
def Sum_averages(array):
    #建立总集合
    subsets=[]
    total_avg = 0
```

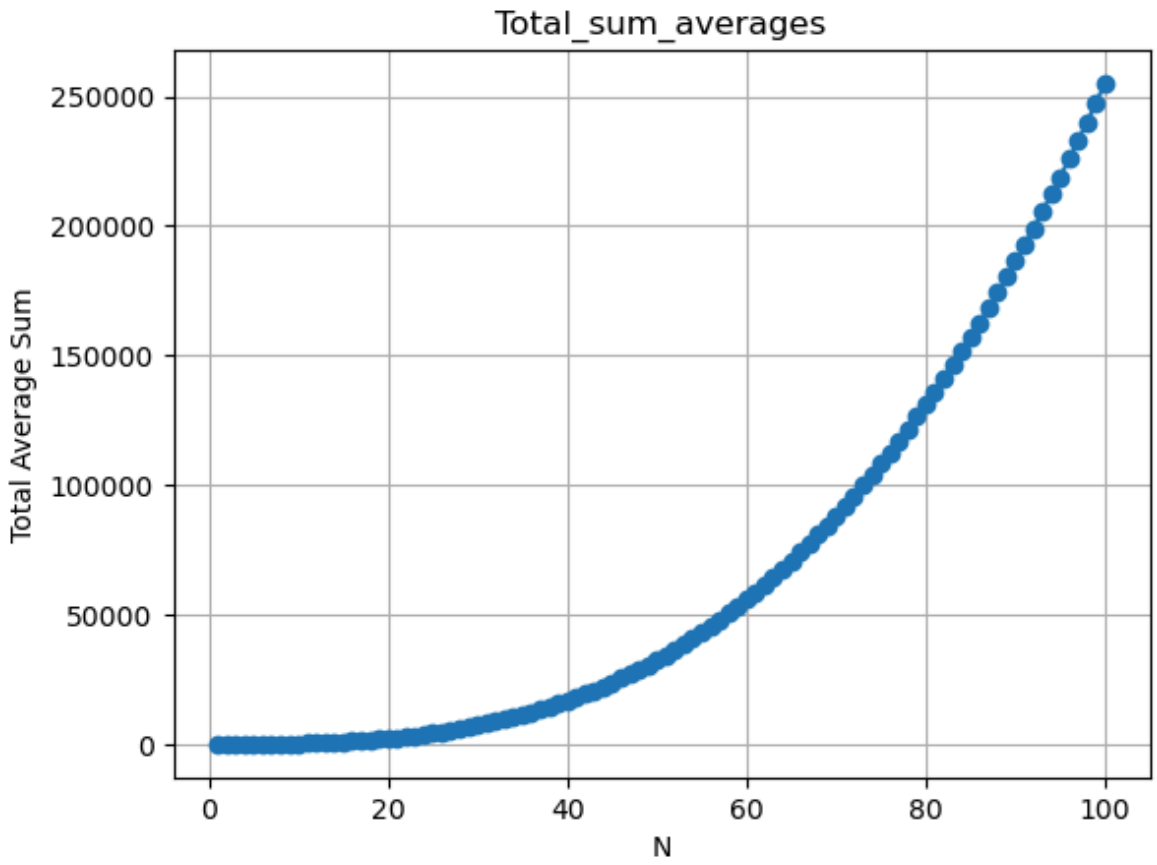
```
    n = len(array)
    # 生成非空子集
    for i in range(1, n+1):  #1~n
        for j in range(n):
            if j+i<=n:  # 从索引j开始取长度为i的子集
                subset=array[j:j+i]   #长度为i的子集
                subsets.append(subset)
                # 计算平均值
                avg=sum(subset)/len(subset)
                total_avg+=avg

    return subsets, total_avg
#使用
example = [1, 2, 3]
print("结果是"+str(Sum_averages(example)))
```

结果是([[1], [2], [3], [1, 2], [2, 3], [1, 2, 3]], 12.0)

4.3 [5 points] Call `Sum_averages` with `N` increasing from `1` to `100`, assign the output to a list called `Total_sum_averages`. Plot `Total_sum_averages`, describe what you see.

In [17]:
```
# 计算从 1 到 100 的平均值总和
Total_sum_averages = []
for N in range(1, 101):#1~100
    #创建数组
    lit = list(range(1, N + 1))
    subsets,total_avg=Sum_averages(lit)
    Total_sum_averages.append(total_avg)
# 绘图
import matplotlib.pyplot as plt
plt.plot(range(1, 101), Total_sum_averages, marker='o',linestyle='-')
plt.title("Total_sum_averages")
plt.xlabel("N")
plt.ylabel("Total Average Sum")
plt.grid(True)
plt.show()
```



Describe what you see. [一条递增的曲线]

[4.1 利用循环N次生成一个含N个元素的列表； 4.2 先生成子集，再计算平均值,输出为：结果是([[1], [2], [3], [1, 2], [2, 3], [1, 2, 3]], 12.0) 4.3 先进行计算，再绘图，最后生成一个递增的曲线]

5. Path counting

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

In [31]:
```
import numpy as np
#Create a matrix with N rows and M columns
def create_matrix(N,M):
    #全为0
    matrix = np.zeros((N,M), dtype=int)
    #fill the right-bottom corner and top-left corner cells with 1
    matrix[0,0] = 1  # 左上角
    matrix[N-1,M-1] = 1  # 右下角
    #其余部分随机填充
    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  # 跳过已经填充的角落
            matrix[i,j] = np.random.randint(0,2)  # 随机填充0或1
    return matrix
#使用
```

```
result_matrix=create_matrix(int(input("行数N=")), int(input("列数M=")))
print(result_matrix)
```

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

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 [33]:
```
def Count_path(matrix,x=0,y=0):
    N,M=matrix.shape
    #走不通，返回0
    if x>=N or y>=M or matrix[x,y]==0:
        return 0
    #到达右下角，返回1
    if x==N-1 and y==M-1:
        return 1
    # 继续向下和向右移动
    return Count_path(matrix,x+1,y) + Count_path(matrix,x,y+1)
#计算路径总数
result=Count_path(result_matrix)
print("路径总数:", result)
```

```
路径总数: 0
```

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 [51]:
```
#运行Count_path1000次
N = 10
M = 8
results = []
#run Count_path for 1000 times
for i in range(1000):
    result_matrix = create_matrix(N,M)  #each time the matrix
    path_new = Count_path(result_matrix)  #计算路径总数
    results.append(path_new)  # 存储结果
total_number=sum(results)
#结果
print("路径总数列表:", results)
print("路径总数:", total_number)
```

```
路径总数列表：[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, 8, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 23,
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, 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, 2, 0, 0, 0, 12, 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, 6, 0, 0, 0, 0, 0, 0, 0, 5, 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, 0, 0, 0, 0, 0, 0, 0, 0, 2, 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, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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,
15, 0, 0, 0, 0, 4, 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, 5, 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, 0, 0, 248, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 5, 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, 9, 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, 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, 0, 0, 4, 0]
路径总数：384
```

Report the mean of total number of paths from the `1000` runs. [是个随机数]

[5.1 先完成指定填充部分，剩下的用遍历随机填充0或1； 5.2 当路走不通时，返回0，若能到达返回1； 5.3 随机遍历1000次，用列表储存结果，最后计算列表总和]