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 [3]:
```
#
def Print_values(a, b, c):
    list1 = sorted([a,b,c],reverse = True) #创建list,并按照从大到小的顺序排序
    print(list1)
    if b == list1[0]:
        print("程序退出")
    else:
        print(list1[0]+list1[1]-10*list1[2])   #将list中的三个数进行计算并输出结果
#Your code here
Print_values(10, 5, 1)
```

```
[10, 5, 1]
5
```

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

[1.函数的入口参数为a,b,c类型为int型 2. 将a,b,c放入list1，并从大到小排序 3. 如果list1[0]是b，说明b最大，根据流程图，结束程序；其他的情况就代入公式里计算出 结果：输出是5]是5

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 [12]:
```
import math
import numpy as np
# def calculate_F_values(N_list):
def calculate_F_values(N_list):
    F_value = []
    for i in N_list:
        F_value.append(F(i))
    print(F_value)
    return F_value
#
def F(x):
    if x == 1:
        return 1
    else:
        return (F(math.ceil(x/3))+2*x)
# Your code here

#测试
N_list = [25,33,2,5]
calculate_F_values(N_list)
```

```
[75, 101, 5, 15]
```

Out[12]:
```
[75, 101, 5, 15]
```

[1.调用random库中的randint，在1-99中随机生成20个正整数组成list2。 2. 定义函数F（x），返回=1，x不等于1时，返回F(ceil(x/3)) + 2x。即采用递归的方式来计算函数值 3. 在函数calculate_F_values(N_list)中遍历N_list中的元素，输出F（x），放入列表F_value中，打印并返回 结果：对于N_list = [25,33,2,5]，输出为[75, 101, 5, 15]

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 [7]:
```
# def find_number_of_ways(sum_x):
#
def Find_number_of_ways(sum_x):
    n = 0
    for a in range(11):                  # a是10个筛子中摇到6的筛子数
        for b in range(11):                  # b是10个筛子中摇到5的筛子数
            for c in range(11):                  # c是10个筛子中摇到4的筛子数
                for d in range(11):                  # d是10个筛子中摇到3的筛子数
                    for e in range(11):                  # e是10个筛子中摇到2的筛子数
                        for f in range(11):                  # f是10个筛子中摇到1的筛子数
                                                                #筛子数之和必须是10，而且点数
                            if (a+b+c+d+e+f == 10) and (6*a + 5*b + 4*c + 3*d + 2*
                                n += 1
    return(n)
# Your code here
```

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 [9]:
```
# Your code here
Number_of_ways = []
for i in range(10, 61):                          # 让i遍历10到60
#     print(i)                                    # 主要是想看一眼代码进度
    Number_of_ways.append(Find_number_of_ways(i))  # 将F(i)的每一个值加到列表中
index = Number_of_ways.index(max(Number_of_ways))  # index找到列表中最大的数的位置
print(f"Waiting for exits.")                      # 把那个位置＋10就是i的值
print(f"The sum that yields the maximum number of ways is {index+10} with {max(Numbe
# print(f"The sum that yields the maximum number of ways is {max_sum} with {max_ways}
```

34得到的最大

```
The sum that yields the maximum number of ways is 34 with 141 ways.
```

So which `x` yields the maximum of `Number_of_ways`? [34]

[把这个问题变成一个加权计算的问题，设有a个筛子抛到点数6，b个筛子抛到点数5，c个筛子抛到点数4，d个筛子抛到点数3，e个筛子抛到点数2, f个筛子抛到点数1，a+b+c+d+e+f = 10而且让a,b,c,d,e,f遍历1到10，6a+5b+4c+3d+2e+f==x没成立一次，n就加1。 然后让i遍历10到60，将Find_number_of_ways(i)放到一个列表中，找到列表中最大值对应的i。结果得出是34。得出结果是34的筛子有141种投法。]

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`.

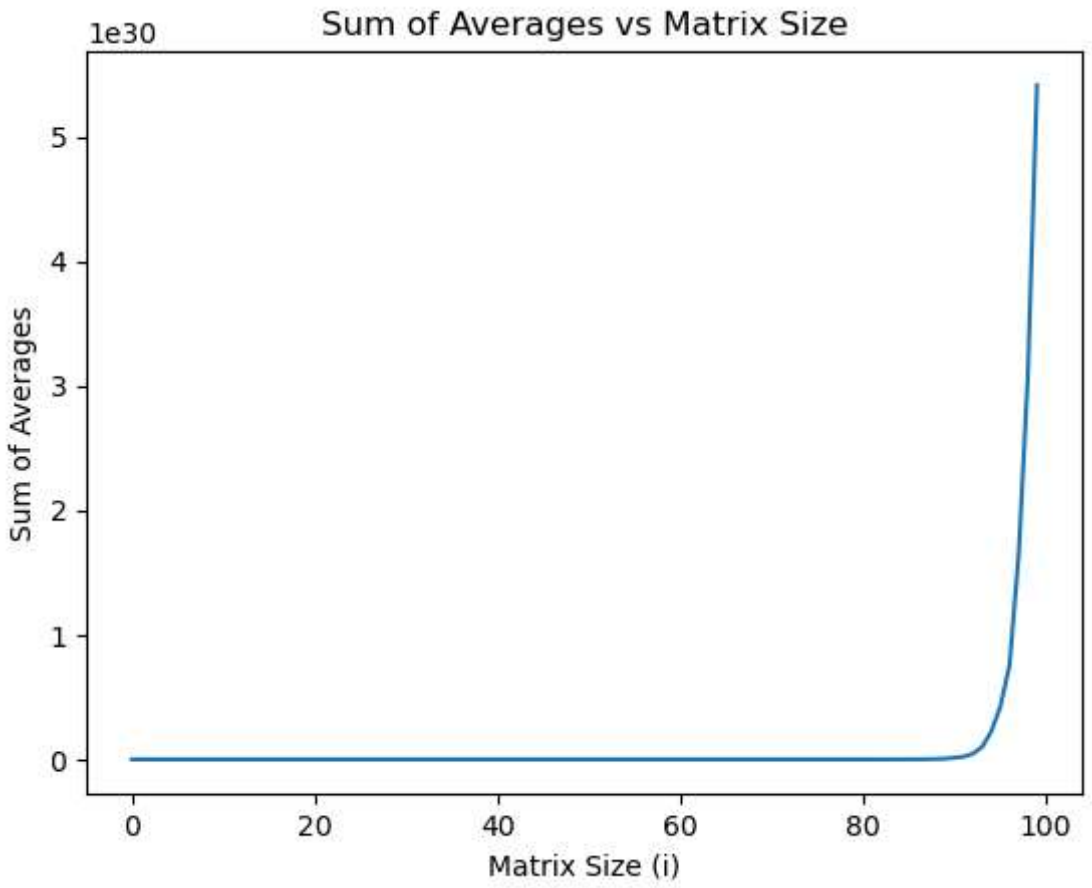
In [2]:
```
import random
# 4.1
# def Random_integer(N):
#     # Your code here
# 4.1写一个函数 Random_integer
def  Random_integer(N):
    Matrix = np.ones(N)                          # 初始化一个N个值为1组成的数组
    for i in range(N):                           # 循环 N 次，给数组的每个元素赋值为 0 至
        Matrix[i] = random.randint(0, 10)
    return Matrix
```

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 [4]:
```
# 4.2
# def Sum_averages(array):
import numpy as np
import math
def Sum_averages(s):
    n = len(s)  # 列表的大小
    sum = 0  # 平均值的总和
    # 遍历集合中的每个元素
    for element in s:
        element_sum = 0  # 当前元素的值占比
        # 对于每一个元素，计算它在每个子集大小中的贡献
        for k in range(1, n + 1):
            # 计算该元素在大小为k的子集中的贡献
            comb_count = math.comb(n - 1, k - 1)  # 从 n-1 个其他元素中选 k-1 个组合
            element_sum += (element / k) * comb_count  #计算组合数之后乘以元素对应该
        # 累加每个元素的值
        sum += element_sum
    return sum
#     # Your code here
```

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 [6]:
```
# 4.3
import matplotlib.pyplot as plt
list1 = []
for i in range(1, 101):
    # print(i)
    result = Sum_averages(Random_integer(i))
    list1.append(result)  # 将结果添加到 list1 中
# print(list1)
# 使用 Matplotlib 绘制 list1 的结果
plt.plot(list1)
plt.xlabel('Matrix Size (i)')
plt.ylabel('Sum of Averages')
plt.title('Sum of Averages vs Matrix Size')
plt.show()
#Your code here
```



1.由上图可以看出，是一个上升曲线，并且在数列长度为90多的时候呈现爆炸式上升的趋势。可能的原因是数组的非空子集个数是2的n次方-1，所以所有的子集的平均值加和也会呈现出指数级增长的过程。 2.图中数列长度在1-90的过程中虽然很平滑，但通过推想，其实不应该是平滑上升的，实际上可能是有起伏的，但是因为纵坐标范围太大了（10的30次方）所以微弱的起伏忽略不计了。

[思路：1.我一开始采用了分别求出每一个子集的平均值的方法，然后发现那个方法计算量太大了，4.3根本算不完 2.我请教了一下张柏欣同学，他说让我试着往找到规律公式的思路上想，然后我就举了几组，试着推理了一下，找到了经验公式，就是数组里的每一个元素对均值总和计算的贡献，应该是这个元素×comb(n - 1, k - 1)/k (其中n是数组的总元素数，k是某个元素跟其他元素组成的子集的元素数）].

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 [10]:
```
# Your code here
import numpy as np
import random
# 5.1创建一个 N 行 M 列的矩阵
def GenerateMatrix(N, M):
    Matrix = np.ones((N, M))           # 初始化一个全为1的矩阵
    for i in range(N):
        for j in range(M):
            Matrix[i, j] = random.randint(0, 1)     #矩阵里的数全部随机赋值0或1
    Matrix[0, 0] = 1                                #矩阵左上角=1
    Matrix[N-1, M-1] = 1                            #矩阵右下角=1
    return Matrix
```

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 [12]:
```
# def Count_path(matrix):
def Count_path(Matrix):
    N, M = Matrix.shape
    for i in range(N):
        for j in range(M):
            if int(Matrix[i, j]) == 1:
                if i == 0 and j == 0:          #对于左上角的元素无需操作
                    continue
                if i == 0:                     #对于第一行的元素，只考虑其左边的元素
                    Matrix[i, j] = Matrix[i, j - 1]
                elif j == 0:                   #对于第一列的元素，只考虑其上边的元素
                    Matrix[i, j] = Matrix[i-1, j ]
                else:                          #其他地方的元素，考虑其上边加左边的两
                    Matrix[i, j] = Matrix[i - 1, j] + Matrix[i, j - 1]
                    # print(Matrix)
    return Matrix[N - 1, M - 1]                #返回，此时右下角的值就是路径方式的总
    # Your code here
```

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 [14]:
```
# Your code here
sum = 0
for i in range(1000):
    sum += Count_path(GenerateMatrix(10, 8))
print(f"The mean of total number of paths from the 1000 runs is {sum/1000}")
```

```
The mean of total number of paths from the 1000 runs is 0.382
```

Report the mean of total number of paths from the 1000 runs. [0.382]

[思路：对于某个位置的元素，走到这个元素有几条路，取决于其上面和左边的两个元素是什么，是1就表一条能走通的路，是0就代表不能走通，一次类推，左边的值加上上边的值就等于能走通的路的数量，以此类推，可以从左上角开始，把每一个数用其左边和上边的值的加和表示，这样一直遍历到右下角，右下角的值就是从左上到右下能走通的路的数量。 结果：运行了几次发现结果不尽相同，但都小于1，是0.几。]