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 [ ]:
```
def Print_values(a,b,c):
    if a > b:
        if b > c:
            result = a+b-10*c
            print(result)
        else:
            if a > c:
                result = a+c-10*b
                print(result)
            else:
                result = c+a-10*b
                print(result)
    else:
        if b > c:
            print("stop")
        else:
            result = c+b-10*a
            print(result)

 # 测试
a = 10
b = 5
c = 1
output = Print_values(a, b, c)
```

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

[我根据观看b站"Python官方课程"的视频巩固知识并完成了这道题。]

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 [1]:
```
import math

def F(x):
    if x==1:
        return 1
    else:
        return F(math.ceil(x/3))+2*x

def continuous_ceiling_function(values):
    results = []
    for x in values:
        results.append(F(x))
    return results

# 测试
test_values = [5, 6, 8,  10]
results = continuous_ceiling_function(test_values)
print(results)
```

```
[15, 17, 23, 33]
```

[通过CSDN博客学习了递归的概念，函数包含了对其自身的调用，该函数就是递归的。递归通常需要一个开始时使用的种子值，向函数传递参数，这个函数是非递归的，题目中的F（1）=1就是入口函数。]

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 [2]:
```
# def find_number_of_ways(sum_x):
def Find_number_of_ways(num_dice, target_sum):

    if target_sum < 0:
        return 0  # 如果目标和小于0, 返回0
    if num_dice == 0:
        return 1 if target_sum == 0 else 0  # 如果没有骰子, 只有目标和也为0时返回

    total_ways = 0  # 初始化组合总数

    for face_value in range(1, 7):
        # 递归调用, 尝试每个面值
        total_ways += Find_number_of_ways(num_dice - 1, target_sum - face_value)

    return total_ways

num_dice = 10

# 测试
target_sum=10
ways = Find_number_of_ways(num_dice, target_sum)
print(f"得到和为 {target_sum} 的方式数: {ways}")
```

```
得到和为 10 的方式数: 1
```

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 [11]:
```
# 设置骰子的数量
num_dice = 10

# 初始化一个列表来存储每个目标和的组合方式数
Number_of_ways = []

# 计算从 10 到 60 的每个目标和的组合方式数
for target_sum in range(10, 61):
    ways = count_ways(num_dice, target_sum)
    Number_of_ways.append(ways)

# 找到最大组合方式数及其对应的目标和
max_ways = max(Number_of_ways)
max_index = Number_of_ways.index(max_ways) + 10  # +10 是因为目标和从 10 开始

print(f"Number_of_ways: {Number_of_ways}")
print(f"最大组合方式数: {max_ways}, 对应的目标和: {max_index}")

# print(f"The sum that yields the maximum number of ways is {max_sum} with {max_
```

```
Number_of_ways: [1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228, 147
940, 243925, 383470, 576565, 831204, 1151370, 1535040, 1972630, 2446300, 2930455,
3393610, 3801535, 4121260, 4325310, 4395456, 4325310, 4121260, 3801535, 3393610,
2930455, 2446300, 1972630, 1535040, 1151370, 831204, 576565, 383470, 243925, 1479
40, 85228, 46420, 23760, 11340, 4995, 2002, 715, 220, 55, 10, 1]
最大组合方式数: 4395456, 对应的目标和: 35
```

So which `x` yields the maximum of `Number_of_ways`? [35]

[这里使用了GPT厘清题目的思路。]

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 [1]:
```
import random
def Random_integer(N):
    # 生成一个包含 N 个随机整数的列表，范围从 0 到 10
    random_array = [random.randint(0, 10) for i in range(N)]
    return random_array

# 测试函数
N = 6  # 示例长度
result = Random_integer(N)
print(f"随机生成的数组: {result}")
```

```
随机生成的数组: [4, 10, 5, 10, 1, 5]
```

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 [2]:
```
from itertools import combinations
#导入 combinations: 使用 itertools.combinations 来生成所有可能的非空子集
def Sum_averages(array):
    total_sum = 0
    n = len(array)

    # 生成所有非空子集
    for i in range(1, n + 1):
        for subset in combinations(array, i):
            average = sum(subset) / len(subset)
            total_sum += average

    return total_sum

# 测试
array = [1, 2]
result = Sum_averages(array)
print(f"The sum of the averages of all subsets is: {result}")
```

```
The sum of the averages of all subsets is: 4.5
```

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.

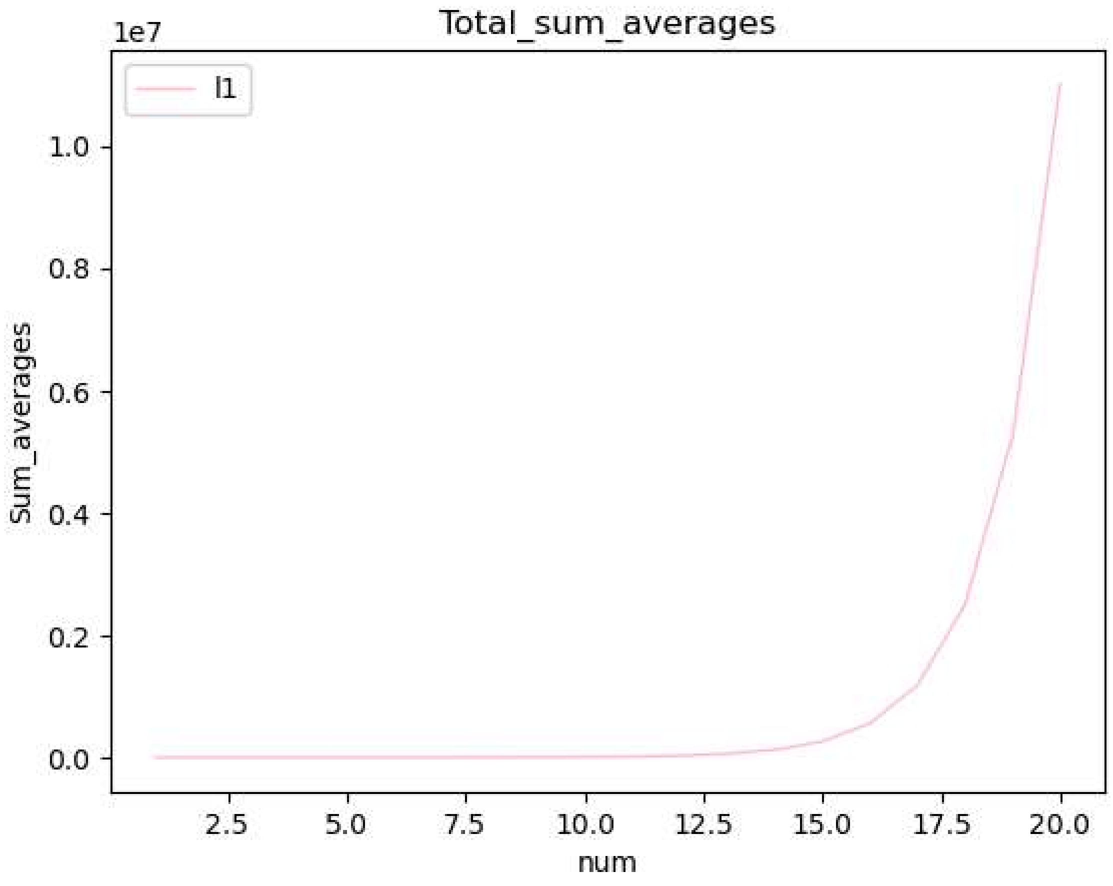
In [3]:
```
import matplotlib.pyplot as plt
import numpy as np

x1 = list(range(1, 21))
```

```
y1 = [Sum_averages(list(range(1, i + 1))) for i in x1]

 # plt.plot(横坐标,纵坐标, 颜色, 线宽)
plt.plot(x1, y1, label = 'l1', color = 'pink', linewidth = 1 )

plt.legend() # 显示标签
plt.title('Total_sum_averages') # 设置图片标题
plt.xlabel("num") # 横轴名字
plt.ylabel("Sum_averages") # 纵轴名字
plt.show()
```



Describe what you see. [初期增长缓慢，趋近于0，随后出现了明显的指数型上升。]

[通过网页搜索如何得到一个集合所产生的所有非空子集，这里调用了itertools.combinations。除此之外，通过CSDN博客和林嘉意同学的帮助，我明白了4.3画图的方法。由于range（1，100）太大，电脑运行不出，以range（1，20）为例。]

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 [2]:
```
import numpy as np

def create_matrix(N, M):
    matrix = np.random.randint(2, size=(N, M))
    matrix[0][0] = 1
    matrix[N-1][M-1] = 1

    return matrix

# 测试
N = 10
M = 8
matrix = create_matrix(N, M)
print(matrix)
```

```
[[1 1 0 1 0 1 0 1]
 [0 0 0 1 1 0 1 1]
 [1 1 0 0 0 1 0 1]
 [0 1 0 1 1 1 1 0]
 [0 1 1 0 0 1 0 0]
 [0 0 1 1 0 1 1 0]
 [1 1 1 1 1 0 1 1]
 [0 1 0 1 1 0 0 1]
 [0 0 0 0 0 0 1 0]
 [1 0 1 0 1 0 1 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 [3]:

```
#5.2
def Count_paths(matrix):
# 计算矩阵的行数 N 和列数 M
    N = len(matrix)
    M = len(matrix[0])

# 创建一个与输入矩阵同样大小的二维数组dp，用于存储到达每个单元格的路径数量，初始值
    dp = np.zeros((N, M), dtype=int)

# 将起始单元格的路径数量设为1，因为从起始点到起始点只有一种方式。
    dp[0][0] = 1

# 双重循环
    for i in range(N):
        for j in range(M):
            if matrix[i][j] == 0:
                dp[i][j] = 0       # 单元格阻塞，没有路径
            else:
                if i > 0:
                    dp[i][j] += dp[i-1][j]  # 从上方单元格的路径
                if j > 0:
                    dp[i][j] += dp[i][j-1]  # 从左侧单元格的路径

# 返回右下角单元格的路径数量，即从左上角到右下角的总路径数。
    return dp[N-1][M-1]

# 测试
total_paths = Count_paths(matrix)
print(total_paths)
```

```
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 [6]:
```
def run_experiment(N, M, runs=1000):
    total_paths = []

    for i in range(runs):
        matrix = create_matrix(N, M)
        paths = Count_paths(matrix)
        total_paths.append(paths)

    # Compute the mean of the total number of paths
    mean_paths = np.mean(total_paths)
    return mean_paths

# 测试
mean_paths = run_experiment(10, 8, 1000)
print(mean_paths)
```

```
0.281
```

Report the mean of total number of paths from the `1000` runs. [0.281]

[通过GPT和何兆轩同学（非本班）的帮助，厘清了5.2题和5.3题解决问题的思路。]