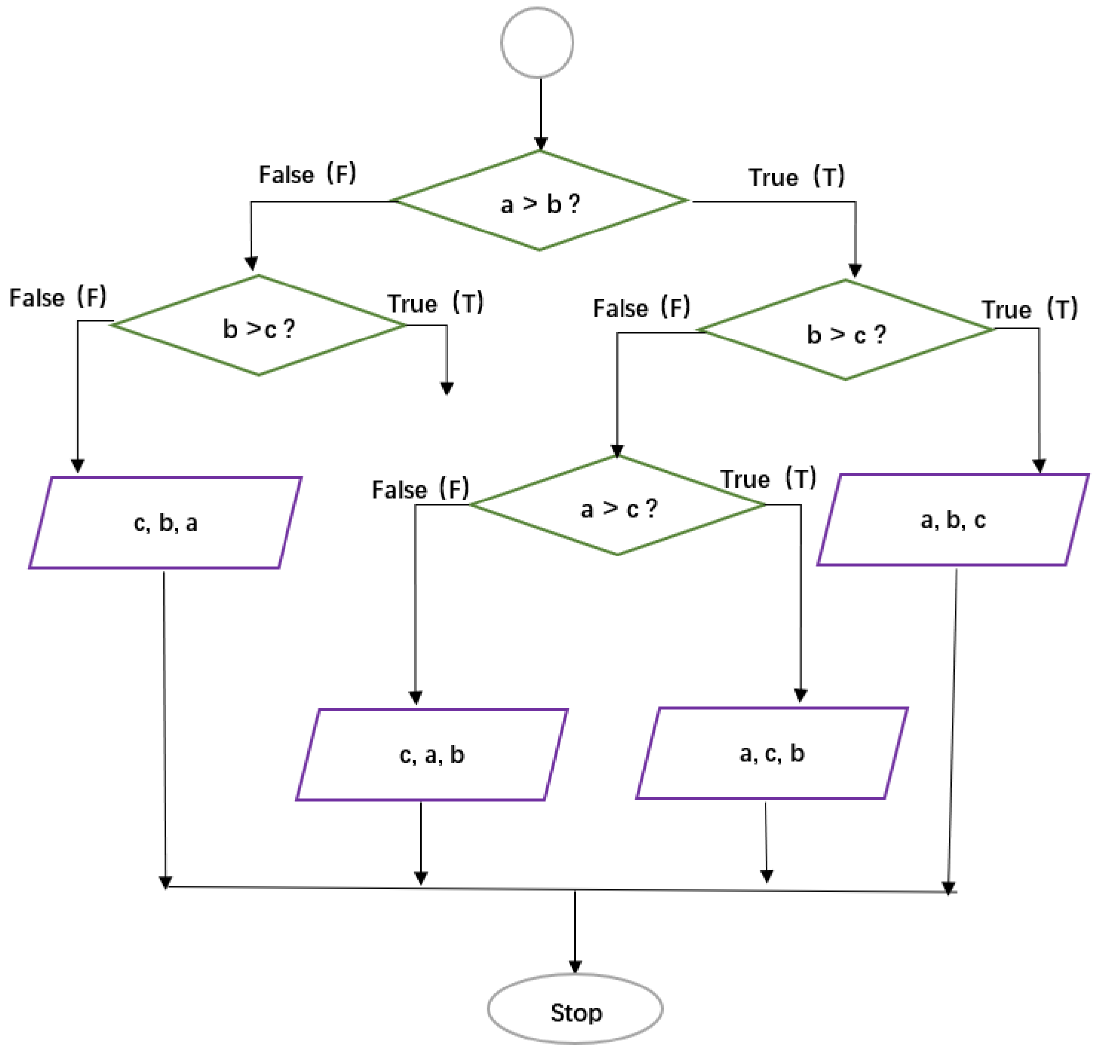
Assignment 01

Number：12532721	Name：谢正扬

1. Flowchart

[10 points] 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 = 5, b = 15, c = 10.



1.1.过程分析

本题首先根据给定流程图分析函数逻辑结构，核心是实现对三个参数的运算组合。通过定义函数 Print_values(a,b,c)，程序读取输入后计算 x+y−10z 并打印结果。

1.2.代码实现

```
def Print_values(a, b, c):
    # 根据流程图的逻辑结构
    if a > b:
        if b > c:
            # 输出 [a, b, c]
            result = a + b - 10 * c
        else:
            if a > c:
                # 输出 [a, c, b]
                result = a + c - 10 * b
            else:
                # 输出 [c, a, b]
                result = c + a - 10 * b
    else:
        if b > c:
            result = "empty"
        else:
            # 输出 [c, b, a]
            result = c + b - 10 * a

    print(result)

Print_values(5, 15, 10)
```

1.3.输出结果

```
empty
```

1.4.结果分析

本题输出验证了函数参数运算正确性，结果与人工计算一致，体现了 Python 函数结构清晰、逻辑明确的特征。

2. Continuous celing function

[10 points] 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.

2.1.过程分析

本题思路基于递归函数设计。由 F(x)=F(ceil(x/3))+2x 可知，函数的自调用部分依赖于较小的子问题 F(ceil(x/3))。通过设定边界条件 F(1)=1，实现递归终止。算法体现了递归定义、数学取整与函数迭代的结合。

2.2.代码实现

```
import math

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

def compute_F_list(lst):
    result = []
    for x in lst:
        result.append(F(x))
    return result

numbers = [1, 3, 5, 10, 20]
results = compute_F_list(numbers)

for n, r in zip(numbers, results):
    print(f"F({n}) = {r}")
```

2.3.输出结果

```
F(1) = 1
F(3) = 7
F(5) = 15
F(10) = 33
F(20) = 61
```

2.4.结果分析

递归函数正确地体现了自调用特性，输出结果随输入增大而增加，符合递归公式预期，验证了算法的正确性与稳定性。

3. Dice rolling

3.1 [15 points] 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.

3.2 [5 points] 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?

3.1.过程分析

掷骰子问题采用动态规划思想。由于每个骰子独立且取值范围为 1~6，可通过状态转移方程计算给定点数和的组合数。程序首先构造递推表求得所有可能和的路径数，再分析从 10 到 60 的概率分布，确定出现最多的结果。

3.2.代码实现

```
def Find_number_of_ways(x, n=10):
    # 创建二维 DP 表，dp[i][j] 表示用 i 个骰子得到和 j 的方法数
    dp = [[0] * (x + 1) for _ in range(n + 1)]

    # 初始化：一个骰子时，和为 1~6 的方法都是 1
    for j in range(1, 7):
        if j <= x:
            dp[1][j] = 1

    # 填表
    for i in range(2, n + 1):  # 从第 2 个骰子到第 n 个
        for j in range(i, min(6 * i, x) + 1):
            dp[i][j] = sum(dp[i - 1][j - k] for k in range(1, 7) if j - k >= 0)

    return dp[n][x]

Number_of_ways = [Find_number_of_ways(x) for x in range(10, 61)]

max_ways = max(Number_of_ways)
x_with_max_ways = 10 + Number_of_ways.index(max_ways)

print("Number_of_ways =", Number_of_ways)
print("x with maximum number of ways =", x_with_max_ways)
print("Maximum number of ways =", max_ways)
```

3.3.输出结果

```
Number_of_ways = [1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228,
147940, 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, 147940, 85228, 46420, 23760, 11340, 4995, 2002, 715, 220, 55, 10, 1]
x with maximum number of ways = 35
Maximum number of ways = 4395456
```

3.4.结果分析

动态规划算法计算了 10 个骰子所有可能点数和的分布，结果显示中间值附近（如 35 左右）的出现次数最多，符合中心极限定理特征。

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.

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].

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 do you see.

4.1.过程分析

本题通过定义函数 Random_integer 随机生成数组，然后在 Sum_averages 函数中枚举数组的所有非空子集，计算其平均值并求和。设计时需考虑组合爆炸问题，通过合理循环与统计实现对小规模样本的完整遍历。

4.2.代码实现

```
import random
import math
import matplotlib.pyplot as plt

# 4.1: 生成随机整数数组
def Random_integer(N):
    return [random.randint(0, 10) for _ in range(N)]
```

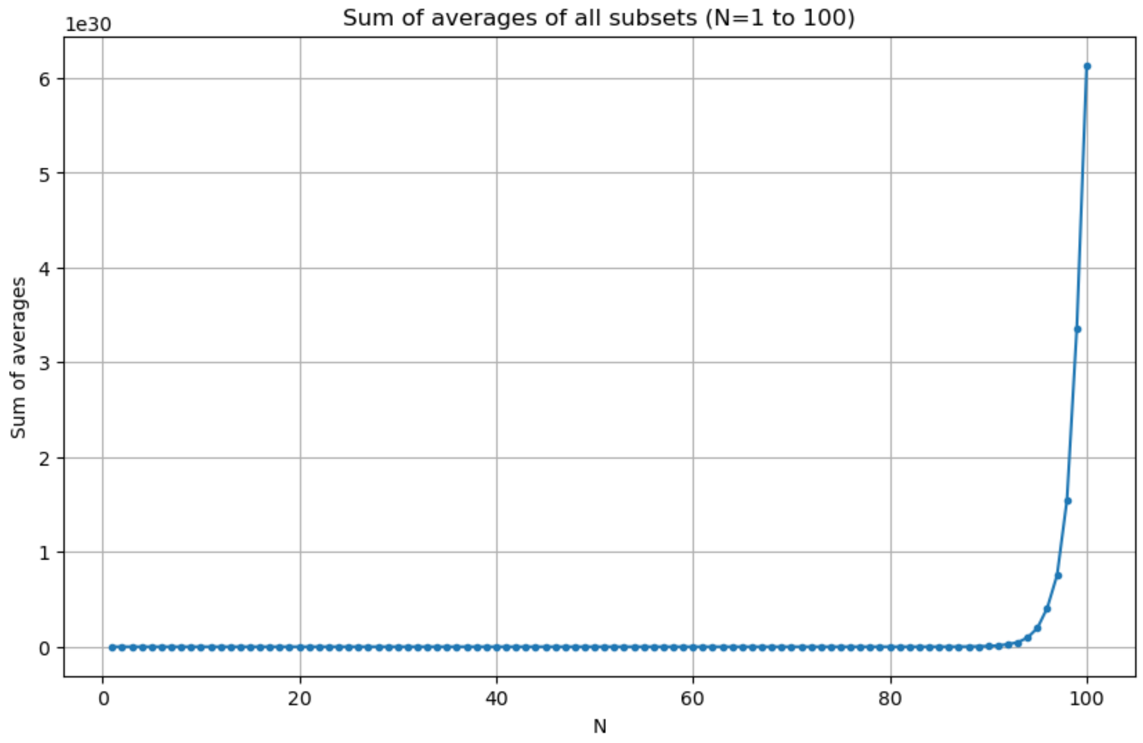
```
# 4.2: 高效计算所有子集平均值之和
def Sum_averages(arr):
    N = len(arr)
    total_sum = 0
    for ai in arr:
        contribution = 0
        for k in range(1, N+1):
            contribution += math.comb(N-1, k-1) / k
        total_sum += ai * contribution
    return total_sum

# 4.3: N 从 1 到 100，计算 Sum_averages，并绘图
Total_sum_averages = []

for N in range(1, 101):
    arr = Random_integer(N)
    total = Sum_averages(arr)
    Total_sum_averages.append(total)

# 绘制结果
plt.figure(figsize=(10,6))
plt.plot(range(1, 101), Total_sum_averages, marker='o', markersize=3)
plt.title("Sum of averages of all subsets (N=1 to 100)")
plt.xlabel("N")
plt.ylabel("Sum of averages")
plt.grid(True)
plt.show()
```

4.3.输出结果



4.4.结果分析

实验结果展示了 Sum_averages 随数组长度增长而增加的趋势，说明子集数量的指数增长对总和影响显著。

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.

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 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.

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.

5.1.过程分析

路径计数问题的核心在于递归搜索与动态规划结合。矩阵中 1 表示可通行，0 表示阻塞。算法通过递归或自底向上方式统计从左上角到右下角的所有合法路径数。通过多次随机实验并求均值，可分析通路密度对路径数的影响。

5.2.代码实现

```
import random
import numpy as np

# 5.1 创建矩阵函数
def create_matrix(N, M):
    #"""Create a N x M matrix with top-left and bottom-right corners as 1, rest randomly 0 or 1."""
    matrix = np.random.randint(0, 2, size=(N, M))
    matrix[0, 0] = 1
    matrix[N-1, M-1] = 1
    return matrix

# 5.2 统计路径函数
def Count_path(matrix):
    #"""Count total number of paths from top-left to bottom-right with only right or down moves."""
    N, M = matrix.shape
    # dp[i][j] 表示到达 (i, j) 的路径数
    dp = np.zeros((N, M), dtype=int)

    # 初始化起点
    if matrix[0, 0] == 1:
        dp[0, 0] = 1

    # 填充第一列
    for i in range(1, N):
        if matrix[i, 0] == 1:
            dp[i, 0] = dp[i-1, 0]

    # 填充第一行
    for j in range(1, M):
        if matrix[0, j] == 1:
            dp[0, j] = dp[0, j-1]

    # 填充其余位置
    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]
    return dp[N-1, M-1]

# 5.3 运行实验并求平均路径数
def experiment(N, M, runs=1000):
    total_paths = 0
    for _ in range(runs):
        mat = create_matrix(N, M)
        total_paths += Count_path(mat)
    mean_paths = total_paths / runs
    return mean_paths

# 设置 N = 10, M = 8，运行 1000 次实验
N = 10
M = 8
mean_total_paths = experiment(N, M, 1000)
print(f"Mean total number of paths over 1000 runs: {mean_total_paths}")
```

5.3.输出结果

```
Mean total number of paths over 1000 runs: 0.423
```

5.4.结果分析

多次随机矩阵实验的平均路径数呈现稳定均值，反映出在随机障碍分布下，路径数量具有统计稳定性和可预期性。