1

对于 Print_values(10, 5, 1) 的调用，执行流程如下：

比较 a 和 b，即 10 > 5，再进入比较 b 和 c，即 5 > 1，为真，执行 result = 10 + 5 - 10 * 1。

计算 result 的值，得到 15 - 10 = 5。

打印 result 的值，即打印 5

2

Compute_values(numbers) 被调用，并传入 numbers = [3, 4, 5, 6, 7, 8, 9, 10] 列表。

对于 numbers 列表中的每个数字，F(x) 会被调用。

F(x) 会根据 x 的值递归地调用自身，直到达到基本情况 x == 1。

每次递归调用都会返回一个值，这个值加上 2 * x 作为结果。

最终，compute_values (numbers) 返回一个包含所有计算结果的列表。

该列表被打印出来[7, 13, 15, 17, 21, 23, 25, 33]

3

初始化一个二维数组 dp，其中 dp[i][j] 表示使用 i 个骰子得到总和 j 的方法数。

第一层循环初始化基本情况：当只有一个骰子时，每个面值（1 到 6）的方法数都是 1。

第二层和第三层循环填充动态规划表。对于每个骰子数 i 和每个可能的总和 j，遍历所有可能的骰子面值 k，并将之前计算的结果累加到当前的总和方法数上。

初始化一个空列表 Number_of_ways。

使用一个循环，计算从 10 到 60 每个总和使用 10 个骰子得到的方法数，并将结果添加到列表中。

使用 max 函数找出列表中的最大值，即最可能的总和。

使用 index 方法找到最大值在列表中的位置，然后加上 10（因为列表索引从 0 开始，而总和从 10 开始）以得到实际的总和值。

打印出哪个总和产生了最多的方法数以及该数目:总和 35 产生了最多的方法数，共有 4395456 种方法

4.1

#调用函数并打印结果，例如填充一个包含 5 个元素的数组[8, 7, 6, 5, 6]

4.2

24

4.3

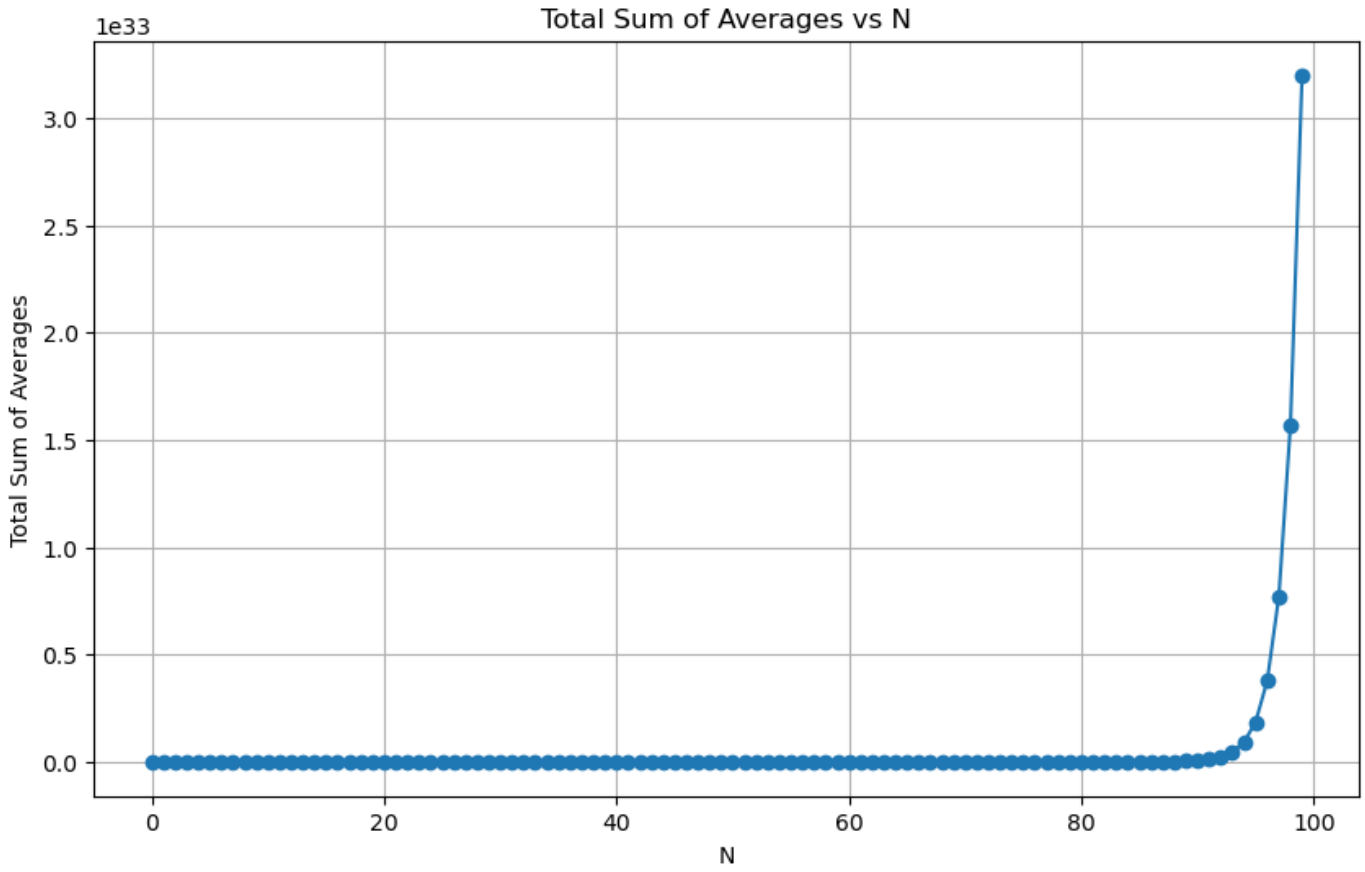
plt.figure(figsize=(10, 6)) 创建了一个新的图形并设置了其大小为 10x6 英寸。

plt.plot(Total_sum_averages, marker='o') 绘制了 Total_sum_averages 的图形，并使用圆圈标记每个数据点。

plt.title, plt.xlabel, 和 plt.ylabel 分别设置了图形的标题和坐标轴标签。

plt.grid(True) 在图形上添加了网格线。

plt.show() 显示了图形。



5.1

创建一个示例矩阵，例如 5 行 4 列

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

5.2-3

计算输入矩阵的行数和列数。

创建一个同样大小的路径计数矩阵 path_count，初始所有元素为 0。

设置左上角的路径计数为 1，如果左上角不是障碍物。

使用两个循环填充第一行和第一列的路径计数。

使用嵌套循环填充剩余部分的路径计数，路径数等于从上方和左方到达当前位置的路径数之和。

返回右下角的路径计数，即从左上角到右下角的总路径数。

设置矩阵的行数 N 为 10、列数 M 为 8 和试验次数 trials 为 1000。

初始化 total_paths 为 0。

进行 1000 次试验，每次试验创建一个矩阵并计算路径数，将结果累加到 total_paths。

计算平均路径数 mean_paths 为 0.379