1.Flowchart

对于不同判定条件，选择对应的计算和输出。主要编写思路为使用 if else 语句，通过判定 False 和 Ture 对应满足的条件，共计五种输出结果：T→T(a, b, c)、T→F→T(a, c, b)、T→F→F(c, a, b)、F→T(Nothing)、F→F(c, b, a)。在输入不同的 a、b、c 值时，根据判定条件输出不同 a、b、c 组合对应 x、y、z，并计算 x+y-10z。

当 a = 10, b = 5, c = 1 时，满足判定顺序 T→T，输出 x=a=10, y=b=5, z=c=1，计算 x+y-10z=5。

2.Continuous ceiling function

这段代码主要定义一个递归函数 F(x)：

首先定义一个字典 memo 用于存储已计算结果，并初始化 x=1 时，F(x)=1；

如果 x 不在 memo 中，即还没有计算过，那么通过递归调用 F(x) = F(ceil(x/3)) + 2x 计算 F(x)，其中 math.ceil(x / 3) 表示将 x 除以 3 并向上取整；

计算完成后，将 F(x) 的值存储在 memo 中，避免重复计算，提高效率。

3.Dice rolling

3.1 该题要求计算使用指定数量的骰子得到特定和的方法数，代码编写思路为：

（1） 首先判断目标和是否在所有骰子可能掷出的范围内，如果不在范围内直接返回 0；

（2）使用动态规划来解决问题，定义一个数组 dp，用于记录得到每个可能和的方法数；

（3）初始化 dp[0] = 1，表示得到和为 0 的唯一方法是没有投掷任何骰子；

（4）对于每一个骰子，使用一个新的数组 next_dp 来记录当前骰子掷出后可能的和以及对应的方法数；

（5）遍历每个可能的和，以及当前骰子的每个可能的面值，更新 next_dp 中的值。

（6）最后将 next_dp 的结果更新到 dp 中，继续下一轮骰子的计算；

（7）最终返回 dp[target_sum]，即为使用所有骰子得到目标和的方法数；

输出结果：The number of ways to get sum 30 with 10 six-faced dice is: 2930455。

3.2 题目中为给出具体对 x 进行的操作，假设统计满足 x 能够被拆分为两个因子之和的不同方式的数量：代码编写思路如下：

（1）计算每个 x 的组合数：`(x - 1) // 2` 计算的是把 x 拆分为两个正整数之和的不同组合数，这样可以快速得到所有结果，并将结果存储在 Number_of_ways

列表中；

（2）找出组合数最多的 x 值：`max_ways = max(Number_of_ways)` 获取 Number_of_ways 列表中的最大值，表示组合数最多的数量；`max_x_values = [10 + i for i, ways in enumerate(Number_of_ways) if ways == max_ways]` 使用列表推导式遍历 Number_of_ways 列表，找到所有组合数等于 max_ways 的位置，并通过 10 + i 来得到对应的 x 值（x 初始值为 10）；

（3）输出结果:

Number_of_ways: [4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11, 11, 12, 12, 13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 23, 24, 24, 25, 25, 26, 26, 27, 27, 28, 28, 29, 29]

x values that yield the maximum number of ways: [59, 60]

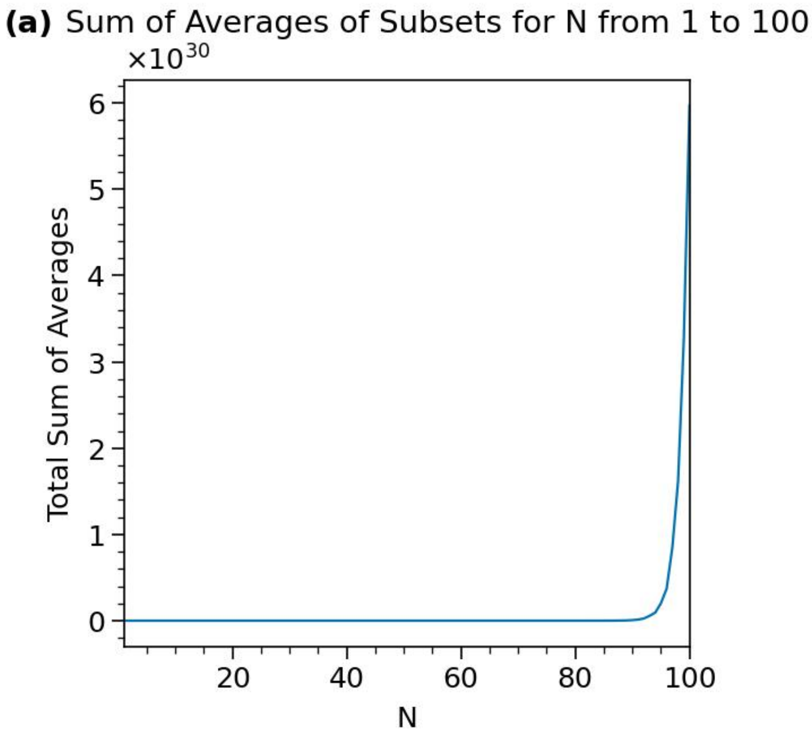
Maximum number of ways: 29

4. Dynamic programming

4.1 生成一个包含 N 个随机整数的数组，这些整数的范围在 0 到 10 之间。编写函数时采用列表推导式：`random.randint(0, 10) for _ in range(N)`，列表推导式中的 `for _ in range(N)` 表示重复执行 N 次，以生成包含 N 个元素的列表。

4.2 计算给定数组的所有非空子集的平均值之和，使用 itertools 库中的 combinations 函数来生成数组 arr 的所有非空子集 `subsets = itertools.combinations(arr, subset_size)`，计算每个子集的平均值并累加求和 `total_sum += np.mean(subset)`。

4.3 调用函数 Sum_averages 计算 N 从 1 到 100 对应的集合的所有非空子集的平均值，结果如图所示：



Total_sum_averages 随着 N 的增大呈指数增加，对于包含 N 个元素的数组，其非空子集的总数为 2^N-1 个。这意味着随着 N 的增加，子集的数量以指数方式增长，Total_sum_averages 随着指数增加。

5. Path counting

5.1 使用 random 模块生成随机矩阵后，指定索引调整元素值。

5.2 函数 `def Count_path(matrix):`用于计算从矩阵左上角到右下角的路径数量，其中只能向下或向右移动，并且只能经过值为 1 的单元格。代码采用动态规划的思想来解决路径计数问题。

（1）创建了一个 dp 动态规划数组，其大小与输入矩阵相同，用于存储到达每个单元格的路径数；

（2）通过双重循环遍历矩阵的每一个单元格，若 `matrix[i, j] == 1`，表示该单元格可以通过：如果该单元格上方的单元格可达（即 i > 0），则将上方单元格的路径数加到当前单元格。如果该单元格左侧的单元格可达（即 j > 0），则将左侧单元格的路径数加到当前单元格。dp[i, j] 中存储的就是到达位置 (i, j) 的所有路径的累加值，最终返回 dp[N-1, M-1] 就得到了从起点到终点的总路径数。

（3）输入参数 N=10，M=8，runs=1000 次，通过 for 循环调用 `create_matrix(N, M)`函数和 `Count_path(matrix)`函数 1000 次，计算 Mean number of paths from 1000 runs: 0.628（使用 random 生成的 matrix，因此每次计算结果可能不同）。