1. Flowchart

当 a = 10, b = 5, c = 1 时，模型的输出有三行，第一行是输入到模型中的列表[10,5,1]，第二行是整理好顺序的列表[10,5,1]，第三行是计算结果 5。

除此以外，还使用了随机生成的数字对模型效果进行验证，其输入及输出如下图所示。

```
Print_values(random.random(), random.random(), random.random())
```

```
[0.04583521846346239, 0.1059965862476614, 0.8891982337922719]
[0.8891982337922719, 0.1059965862476614, 0.04583521846346239]
0.5368426354053094
```

```
Print_values(random.randint(-20,0), random.randint(-20,0), random.randint(-20,0))
```

```
[-4, -11, -13]
[-4, -11, -13]
115
```

```
Print_values(random.randint(0,100), random.randint(0,100), random.randint(0,100))
```

```
[33, 87, 89]
[89, 87, 33]
-154
```

2. Continuous ceiling function

题目要求输入到函数中的数字必须是正整数，因此在代码中应该体现出来，若输入的列表中出现负数或是非整数，应该排除掉，对此功能的测试以及测试结果如下图所示。

```
list_2_1 = [5,8,47,10]
Continuous_ceiling(list_2_1)
list_2_2 = [5,8,-47,10]
Continuous_ceiling(list_2_2)
list_2_3 = [5,8,7.9,10]
Continuous_ceiling(list_2_2)
```

```
[15, 23, 143, 33]
[15, 23, 'NA', 33]
[15, 23, 'NA', 33]
```

3. Dice rolling

在这题里面，要考虑的主要是组合和组合之间的重复问题，骰子之间是不存在区别的，因此组合[1,2,3]和组合[2,1,3]代表的是同一种情况。在这一情况下，我们可以将“十个骰子掷出后，得到的结果是怎样的”这一问题转化为“十个骰子中，1 到 6 这六个数字分别出现了几次”。这样一来，我们就可以不重复地列出所有的组合，计算这些组合的值，并在这些值里面找出特定值出现的次数。

当把所有可能的组合列出来后，哪个和对应的组合可能性最多也是一目了然的。因此，当和为 34、35 或 36 时，对应的组合数最多。皆为 141。

4. Dynamic programming

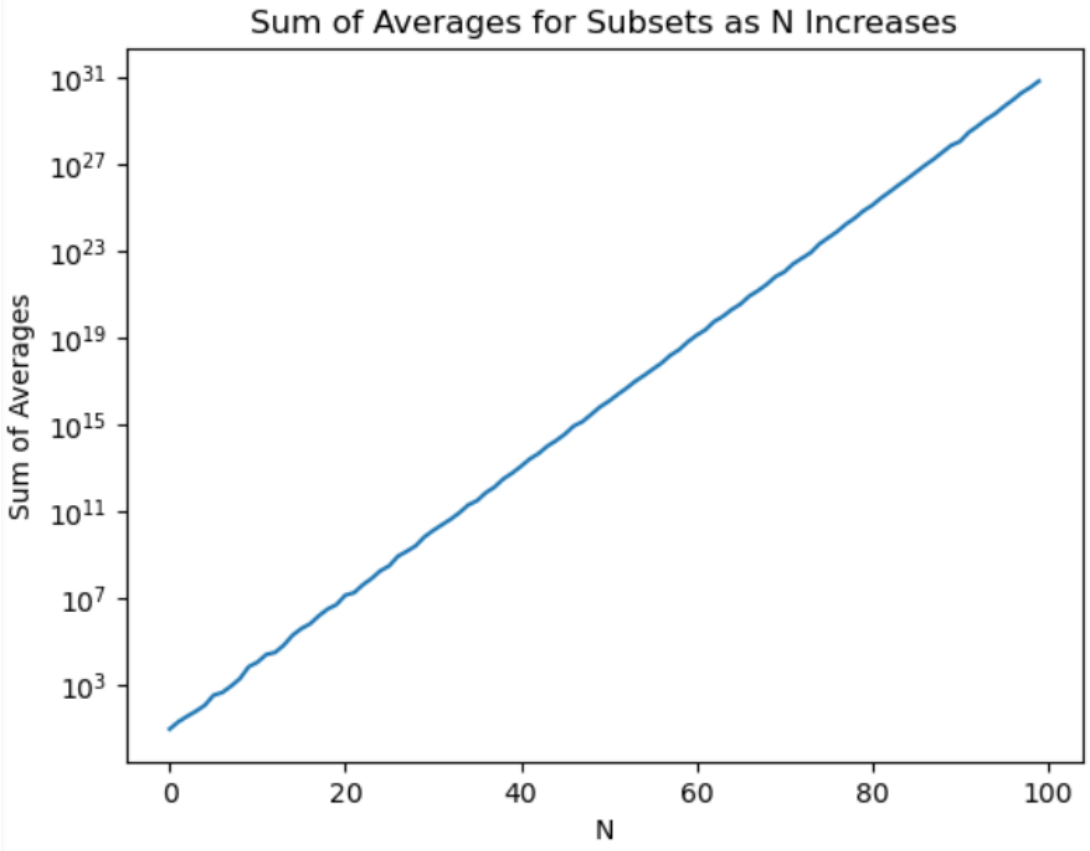
做这道题的时候要在最开始就考虑到第三问里面的要求，第三问要计算包含 100 个元素的集合中所有子集的平均值之和。一个拥有 100 个元素的集合可以划分出 2^{100} 个子集，要把这些子集的平均数一个个列出来再去求和显然是不可能的，因此，需要使用数学方法直接

计算出一个集合中所有子集的平均数之和。

在此情况下，所有子集的平均值之和问题可以转化为“一个元素在所有子集中出现了几次，并且在求平均值的时候被除以几”的问题。因此，对于一个元素数量为 N 的集合中的元素 a 来说，它在元素数量为 X 的子集中出现的次数可以用C_{X-1}^{N-1}来表示，进而元素 a 在所有子集的平均数之和中的贡献可以表达为$a * \sum_{X=1}^{N}(C_{X-1}^{N-1}/X)$。综上可知，对于一个拥有 N 个元素的集合，认为其所有元素之和为 $\text{sum}_{\text{elements}}$，其所有子集的平均数之和如下所示：

$$\text{sum}_{subsets\_avg} = sum_{elements} * \sum_{X=1}^{N}(C_{X-1}^{N-1}/X)$$

最终得到的结果如下图所示，一个集合所有子集的平均值之和随着集合中元素数量 N 的上升，在指数级别上呈现出线性增长的趋势，同时伴随有一定的波动。指数级别上线性增长的出现可能是因为每当 N 增加 1，集合的子集数量就会翻倍。此外，数据中的波动可能是由于随机数生成的元素值在不同位置上的分布不均造成的。当新加入的元素值较大时，它们对子集平均值的提升作用更为显著，这可能导致在某些 N 值下，总平均值之和出现较大的跳跃。相反，当新加入的元素值较小或为 0 时，它们对子集平均值的提升作用较小，这可能导致总平均值之和的增长速度暂时放缓。



5. Path counting

解决这个问题的基本方法是利用动态规划，其中每个单元格的路径数量是由其左侧和上方单元格的路径数量之和决定的。然而，对于第 1 行和第 1 列的单元格，这种计算方式需要特别处理，因为它们只有一条路径可以到达，即从起点开始的路径。在此情况下，当第 1 行或第 1 列中出现 0 时，它会阻断该行或该列中所有后续单元格的路径，因为无法绕过 0 继续向右或向下移动。因此，在计算路径数量时，必须先处理第 1 行和第 1 列，确保任何 0 值都能正确地阻断路径。再处理剩余单元格，这样可以确保路径数量的计算是准确的。

在此规则下，计算出 1000 个的 8 行 10 列矩阵的路径数平均值为 0.337，其中 981 个矩阵中不存在一条可以从左上角通往右下角的路径。