1.

用 random 模块生成了三个介于 0 到 100 之间的随机数（a、b、c），并且使用了 round 函数将这些数四舍五入到最接近的整数，然后判断这三个数的大小关系，并根据这个关系输出不同的结果：
如果 a 大于 b 且 b 大于 c，那么输出"a,b,c"。
如果 a 大于 b，b 小于或等于 c，且 a 大于 c，那么输出"a,c,b"。
如果 a 大于 b，b 小于或等于 c，且 a 小于或等于 c，那么输出"c,a,b"。
如果 a 小于或等于 b 且 b 大于 c，那么输出"please determine the size of a and c "，提示需要确定 a 和 c 的大小关系。
如果 a 小于或等于 b 且 b 小于或等于 c，那么输出"c,b,a"

2.1

生成两个随机整数矩阵 M1 和 M2。
首先，导入 numpy 和 random 两个库。numpy 提供了矩阵运算的功能；random 库用于生成随机数。
然后，用 random.randint(0, 50)函数生成一个介于 0 到 50 之间的随机整数，并用 for 循环生成了 50 个这样的随机整数。这些整数被存储在一个列表中，然后使用 numpy 的 array 函数将这个列表转换为 numpy 数组。
用 numpy 的 reshape 函数将这个一维数组重塑为二维数组。M1 重塑为一个 5 行 10 列的二维数组，M2 重塑为一个 10 行 5 列的二维数组。
最后输出 M1，M2 矩阵。

2.2

定义一个 Matrix_multip(M1,M2)函数，用于计算两个矩阵（M1 和 M2）的乘积，函数 Matrix_multip(M1,M2)定义了如何进行矩阵乘法。在矩阵乘法中，结果矩阵的每个元素都是由原矩阵的一行元素与另一个矩阵的一列元素对应相乘然后相加得到的。
首先定义两个空列表 list1 和 res。list1 用于暂时存储计算结果，res 用于存储最终的矩阵乘法结果。
然后，函数使用两个嵌套的 for 循环遍历 M1 的每一行和 M2 的每一列。在内部循环中，取出 M1 的一行和 M2 的一列，然后将它们对应元素相乘并相加，得到的结果存储在 list1 中。
在外部循环的每一次迭代结束时，将 list1 转换为 numpy 数组并添加到 res 中，然后清空 list1，准备进行下一次迭代。
最后，将 res 转换为 numpy 数组输出。
使用 if name == "main":来确保只有在直接运行这个脚本时才会执行下面的代码。
调用 Matrix_multip(M1, M2)函数，计算之前生成的两个矩阵 M1 和 M2 的乘积，并将结果存储在变量 res 中。

3.

generate_pascals_triangle 函数：这个函数接收一个参数 n，表示要生成的帕斯卡三角的行数。函数首先创建一个只包含一个元素 1 的列表 triangle，然后对于从 1 到 n-1 的每一个 i，生成第 i 行的数字并添加到 triangle 中。每一行的

数字是由上一行的相邻两个数字的和得到的，每一行的第一个和最后一个数字都是 1。
主程序部分：首先定义一个列表 n_values，包含两个元素 100 和 200。然后，对于 n_values 中的每一个 n，调用 generate_pascals_triangle 函数生成帕斯卡三角，并输出每一行的数字。

4.

通过 Least_moves 的递归函数实现：
如果输入的 x 等于 1，那么返回 0，因为不需要任何步骤就可以从 1 降到 1。
如果 x 是偶数，那么 x 除以 2，并且步骤数加 1，然后递归调用 Least_moves 函数处理结果。
如果 x 是奇数，那么 x 减去 1（这样可以确保结果是偶数），并且步骤数加 1，然后递归调用 Least_moves 函数处理结果。
然后，代码中使用了 random.randint(0, 100) 生成一个 0 到 100 之间的随机整数 x，并且输出从 x 到 1 所需要的最少步骤。

5.

5.1

在数字字符串 "123456789" 中插入 '+', '-' 或者不插入任何操作符，寻找所有可能的表达式，使得这些表达式的计算结果等于一个在 1 到 100 之间的随机整数 A。这个过程对于 A 从 1 到 100 的每一个值都会进行，最后将所有的结果存储在字典 results 中。
Find_expression 函数：这个函数接收两个参数，一个整数 A 和一个字符串 nums。它首先定义了一个名为 find_all 的内部函数，这个函数用于生成所有可能的通过在 nums 中插入操作符得到的表达式。然后，Find_expression 函数通过调用 find_all 函数生成所有可能的表达式，并通过 eval 函数计算这些表达式的值，只保留那些值等于 A 的表达式。
主程序部分：首先定义一个空的字典 results，然后对于 1 到 100 的每一个整数 a，调用 Find_expression 函数寻找所有满足条件的表达式，并将结果存储在 results 字典中。itertools.product 函数用于生成所有可能的操作符组合
eval 函数用于计算一个字符串形式的表达式的值。
因为结果比较多，所以我加了 tqdm 模块来显示进度条，方便了解程序运行的进度。

5.2

和 5.1 一样为方便查看进度，使用了 tqdm 模块来显示进度条。
绘制一个图表，显示在前面代码中找到的满足条件的表达式的数量与整数 A 的关系。对于 A 从 1 到 100 的每一个值，计算满足条件的表达式的数量，然后将这些数量与对应的 A 值一起绘制在一个二维图表中。
首先初始化一个全为 NaN 的 numpy 数组 y，长度为 100。这个数组用于存储对于每一个 A 值，满足条件的表达式的数量。
对于 0 到 99 的每一个整数 i，计算满足条件的表达式的数量（即 results[i+1] 的长度），并将结果存储在 y[i] 中。

定义 x 轴的值为从 1 到 100 的整数。
创建一个新的图表，并在这个图表中绘制 y 与 x 的关系。设置 x 轴的刻度和标签，设置 x 轴和 y 轴的标题，以及图表的标题。
最后显示图表即可。
np.full 函数用于创建一个全为给定值的数组
plt.figure 函数用于创建一个新的图表
plt.plot 函数用于在图表中绘制数据
plt.xticks 函数用于设置 x 轴的刻度和标签
plt.xlabel、plt.ylabel 和 plt.title 函数用于设置 x 轴、y 轴和图表的标题
plt.show 函数用于显示图表。