

注：为了使输出结果好看一些，因此我在输出结果的时候都格式化字符串了。

1、首先，它会从用户那里获取三个数字 a, b, c。

通过一系列的 if-elif 语句来检查这三个数字的大小关系。

如果 a 小于等于 b 并且 b 小于等于 c，打印 c, b, a。

如果 b 小于等于 a 并且 a 小于等于 c，打印 c, a, b。

如果 b 小于等于 c 并且 c 小于等于 a，打印 a, c, b。

如果 c 小于等于 b 并且 b 小于等于 a，打印 a, b, c。

如果 c 小于等于 a 并且 a 小于等于 b，打印 b, a, c。

如果 a 小于等于 c 并且 c 小于等于 b，打印 b, c, a。

如果 a, b, 和 c 三者相等，打印 a, b, c。

2、(1) 首先，定义了四个变量分别代表了两个矩阵的行列数。rows_M1 和 cols_M1 分别为第一个矩阵。通过两层列表推导式生成了随机矩阵 M1 和 M2。其中，生成了一个随机整数，范围在 0 到 50 之间。对于 M1，使用了外层的 range(rows_M1) 和内层的 range(cols_M1) 来生成指定行数和列数的矩阵。对于 M2 也是如此。使用两个循环分别打印了矩阵 M1 和 M2。

(2) 定义了一个名为 matrix_multip 的函数，接受两个参数 M1 和 M2，表示要进行乘法运算的两个矩阵。接着，创建了一个用于存放结果的矩阵 result，其行数为第一个矩阵的行数，列数为第二个矩阵的列数。使用三层循环，分别遍历第一个矩阵的行 (i 变量)、第二个矩阵的列 (j 变量) 以及相乘的中间项 (k 变量)，这里实现了矩阵乘法的计算过程。

3、首先定义函数，如果 k=1，直接返回一个列表 [1]，表示杨辉三角形的第一行。如果 k 等于 2，同样返回一个列表 [1, 1]，表示杨辉三角形的前两行。如果 k>2，生成前 k-1 行的杨辉三角形。接下来，创建一个新的列表，并将第一个元素设为 1，表示新行的第一个元素。使用一个循环遍历从 1 到 k-2 的范围（即除去新行的第一个元素和最后一个元素），并将 row[i - 1] 和 row[i] 相加得到新行的中间元素。将中间元素依次添加到新列表。最后，将新行的最后一个元素设为 1，表示新行的最后一个元素。

4、此题参考了 csdn 上的递归算法。

定义 Least_moves 是一个递归函数，一个整数 n 作为输入。如果 n 等于 1，表示已经达到目标状态，返回 0。如果 n 是奇数，表示不能整除 2，只能通过减 1 来进行操作。递归

调用 `Least_moves(n-1)`，并在结果上加 1，表示将 n 缩减为 $n-1$ 需要的最少步骤数。

如果 n 是偶数，可以选择减 1 或者将其除以 2。递归调用 `Least_moves(n-1)` 和 `Least_moves(int(n/2))`，并在两者的结果中取最小值，表示选择最少步骤的路径。

5、此题和师兄进行讨论，建议我使用 `itertools` 函数，因此我通过查阅 [csdn](#) 大量相关资料，以下是 [csdn](#) 上关于 `itertools` 的简介，此题较难，所以写的时候参考大量的资料。

`1. repeat(object[, times]):` 重复生成object，可指定重复次数。

使用 `itertools` 可以方便地处理迭代器操作，提高代码的效率和可读性。Python 中的 `itertools` 模块是一个集成了一些用于迭代器操作的函数的模块。下面是一些 `itertools` 模块的常用函数及其用法：

1. `itertools.chain(*iterables)` 该函数可以把多个可迭代对象拼接成一个迭代器，返回值是一个迭代器。例如：`chain('ABC', 'DEF')` 返回值是一个包含 A、B、C、D、E、F 的迭代器。
2. `itertools.combinations(iterable, r)` 该函数返回一个迭代器，生成由 `iterable` 中所有长度为 `r` 的组合。例如：`combinations('ABCD', 2)` 返回值是一个包含 AB、AC、AD、BC、BD、CD 的迭代器。
3. `itertools.product(*iterables, repeat=1)` 该函数返回一个迭代器，生成由 `iterables` 中的元素的笛卡尔积，`repeat` 参数指定重复迭代的次数。例如：`product('ABCD', repeat=2)` 返回值是一个包含 AA、AB、AC、AD、BA、BB、BC、BD、CA、CB、CC、CD、DA、DB、DC、DD 的迭代器。
4. `itertools.islice(iterable, start, stop[, step])` 该函数返回一个迭代器，生成从 `iterable` 中 `start` 到 `stop-1` 之间的元素，`step` 参数指定步长。例如：`islice('ABCDEFGH', 2, None)` 返回值是一个包含 C、D、E、F、G 的迭代器。

(1) 首先定义目标函数：该函数用于寻找可以得到目标数字的数学表达式。首先，定义了一个嵌套函数 `evaluate(expression)`，用于计算给定表达式的值，如果表达式合法则返回计算结果，否则返回 `None`。然后，生成了包含数字字符 '123456789' 的字符串 `digits`，以及一个包含 `+`、`-`、和空字符串 `' '` 的列表 `add_min`，用于生成表达式的运算符组合。使用 `itertools.product` 生成所有可能的运算符组合，并构建表达式。对每个生成的表达式进行计算，如果其结果等于目标值，则将该表达式添加到 `solutions` 列表中。最终，返回包含所有满足条件的表达式的列表 `solutions`。

(2) 调用了 `Find_Expression(target)` 获取满足条件的表达式列表，并逐个打印出来。随机生成一个整数 `c`，并计算在该随机数下可以得到的表达式数量 `b`。打印出当随机数为 `c` 时，满足条件的表达式，并输出生成的结果数量。

统计从 1 到 100 每个数字可以得到的表达式数量，找到最大值和最小值的数量以及对应的数字。

使用 `Matplotlib` 绘制了一个折线图，横坐标为数字 1 到 99，纵坐标为对应数字能够生成的表达式数量。

图像如下：

