外层循环尝试所有的  $n_1$  值,即 1 到 23。每次迭代  $n_1$  时,内层循环迭代所有的  $n_2$  值,即 1 到(24 $-n_1$ )。对于每个( $n_1, n_2$ )的组合,脚本首先设置  $n_3$  的值为 25-( $n_1+n_2$ ),然后检查三个数的平方和是否为 243。如果满足条件,输出结果并停止脚本。

## 试一试 7-5

创建图 7-16 的脚本并运行,看看  $n_1$ 、 $n_2$  和  $n_3$  是否存在。仔细研究脚本,你会发现有些 ( $n_1$ ,  $n_2$ ) 组合的测试是多余的。例如,在第一轮迭代中,其外层循环将测试组合 (1, 2),而第二轮将测试(2, 1)。显然,第二次测试是没有必要的,我们仅需测试其中之一。只要将内层循环的  $n_2$  从 1 开始修改为从  $n_1$  开始,便能消除多余的测试。完成修改,确保程序依然能正常运行。

## 递归:调用自身的过程

Recursion.sb2

脚本的重复执行除了可以通过循环结构实现外,还可以通过另一个强大的技术实现:递归。所谓递归,是指过程自己直接调用自己(即A调用A),或自己间接调用自己(如A调用B,B调用C,C调用A)。虽然你可能好奇这么做有何意义,但是递归确实能简化大量的计算机科学问题。我们先通过一个简单的案例讲解递归的概念,如图7-17所示。

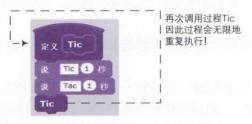


图 7-17: 递归的过程

过程 Tic 执行两个说···积木(先说 Tic (嘀),再说 Tac (嗒)) 后再一次调用自己。当再次调用过程 Tic 时,角色仍然说嘀嗒,然 后再次调用自己。这个过程无穷无尽地执行,直到你单击了绿旗旁 的红色停止按钮。我们并没有使用循环结构也实现了说···积木的重