置标志变量 gotPass 的值，它表示密码验证的结果。当过程返回后，**如果…那么**积木测试标志变量 gotPass，从而决定是否有权限访问系统。如果 gotPass 为 1，意味着用户输入了正确的密码，脚本则使用**说…**积木显示“密码正确！”随后切换造型为 on。否则显示“拒绝访问！”，造型依然是 off。

过程 GetPassword 首先设置标志变量 gotPass 为 0，表示现在还未收到正确的密码。然后初始化变量 failCount 为 0。它表示密码输入错误的次数，也就是本例中的循环计数器。脚本随后**重复执行 3 次**，因为本例中我们认为 3 次是最多的尝试次数。每次迭代前先要求用户输入密码。如果输入正确（本例中为 Pass123），脚本设置标志变量 gotPass 为 1，然后使用**停止当前脚本**积木结束本过程并返回到主脚本。否则，若用户还未用完三次机会，程序会显示一段错误信息，同时再给用户一次机会。若用户连续三次输入错误密码，**重复执行 3 次**结束，过程返回至主脚本，而标志变量 gotPass 的值依然为 0。

**试一试 7-4**

打开并运行程序。若在询问密码时我们输入的密码是 paSS123，而非脚本中标准的 Pass123，用户能否通过认证？这说明字符串的比较有什么特点？尝试使用**重复执行直到**积木实现过程 GetPassword。

## 灵活的循环计数

*CountingByConstAmount.sb2*

循环计数器通常是根据不同的场景和问题灵活变化的。例如，图 7-13 中 ❶ 的循环计数器每次增加 5，实现从 5 到 55 的计数。脚本 ❷ 的循环计数器每次减少 11，从 99 迭代到 0。换言之，每次迭代该循环计数器的值为 99，88，77，…，11，0。


图 7-13：灵活地修改循环计数器