第5章 循环结构程序

如果说分支给了程序选择的能力，那么可以说循环赋予了程序计算的力量。计算机之所以对人类产生了巨大的帮助，一个极其重要的原因是计算机可以高速地做循环，而且是不知疲倦地做循环。

本章介绍循环的基本语法，以及利用循环解决实际问题。

5.1 循环与重复计算

“绳锯木断，水滴石穿”实质上是一个量变到质变的过程，也表明了重复蕴含着巨大的力量，只是往往重复的时间很长很长。类似这样的过程对人类而言枯燥且无趣，但通过计算机的高速性可以大大加速整个过程。

Python 提供了 while 和 for 两种循环控制语句，二者各有所长，都有自己擅长处理的情况。虽然有时二者可以通用，但是遇到具体问题选择合适的循环不仅便于代码实现而且可以降低出错概率。

5.2 while 循环

while 循环非常适合处理未知循环次数的情况。它的语法规则如下：

```
while 条件:
    语句块 1
else:
    语句块 2
```

其中，条件通常称为循环条件，语句块 1 称为循环体。执行 while 循环时，首先判断循环条件是否成立，如果成立就执行循环体，否则结束循环。每次循环体完成后会再次判断循环条件是否成立，然后重复上面的过程。

如果循环是正常结束的，会执行语句块 2，也就是说，如果循环是非正常结束的，不会执行语句块 2。可见语句块 2 最多被执行一次，至少被执行零次。但是要强调一下，while 循环的 else 部分不是必需的，可以省略。

【例 5-1】 利用 while 循环编写程序计算 1～100(含 1 和 100)的所有整数的和。

考虑用一个变量 res 存储整数和，该变量一开始赋值为 0，然后循环 100 次，依次把 1～100 累加到变量 res 中，最后变量 res 就是所要的结果。具体程序如下：

程序 5-1 求 1～100 的整数的和

```
#计算 1～100 的整数的和

res = 0                  #存放累加和
i = 1                    #循环控制变量
while i <= 100:          #循环 100 次
    res += i             #累加
    i = i + 1            #循环控制变量加 1
print("1 + 2 + 3 + … + 100 = ",res).
```

运行结果：

```
1 + 2 + 3 + … + 100 =  5050
```

实际上利用循环求 1～100 的整数的和是比较低效的，因为 1～100 的整数构成了一个公差为 1 的等差数列，所以利用等差数列的求和公式可以无须循环而直接得到结果。此外，基于 Python 自身附带的函数和数据结构，使用如下这样的一句代码也可以解决该问题。

```
print("1 + 2 + 3 + … + 100 = ",sum(range(1,101)))      #利用 sum()函数求累加和
```

在用 while 循环时候，要避免写成死循环。所谓死循环，就是无法结束的循环，主要原因是 while 后面的条件一直成立。例如例 5-1 的程序中如果循环体内忘记把循环控制变量加 1，就会变成死循环。当然死循环也不是一无所用的，工业控制领域的很多程序会故意写成死循环，直到设备被关机为止。

【例 5-2】 编写程序，让计算机产生多道随机数构成的加法口算题，每个运算数为 0～100，让用户输入结果，如果用户做对了，就继续循环，直到用户做错了题目才停止，最后显示用户做对了多少题。

本题需要使用 Python 内置的 random 模块来产生随机数，random 模块提供了多个产生不同随机数的函数，表 5-1 列出了部分常用函数以及说明，这里选用其中的 randint()函数产生 1～10 000 的随机整数。

表 5-1 random 模块中部分常用函数以及说明

函　　数	说　　明
seed(a=None)	初始化随机发生器的种子数，如果 a 省略则使用当前时间
random()	生成一个 0～1 的随机浮点数
randint(a,b)	生成一个 a～b 的随机整数
uniform(a,b)	生成一个 a～b 的随机浮点数
choice(seq)	从序列 seq 中随机抽取一个元素
shuffle(lst)	对列表 lst 中元素随机排序，返回值是 None

为了统计用户做对的题数，可以设置一个初始值为 0 的计数器变量，做对一题后立刻让该变量的值加 1。

此外，本题并不知道用户到底做多少题结束，只要用户输入的数字与两个随机数的和不相等就继续循环，可见是一个未知循环次数的问题。具体程序如下：

程序 5-2 加法口算题

```
#加法口算题

import random
opt1 = random.randint(1,100)
opt2 = random.randint(1,100)
print("{:2d} + {:2d} = ".format(opt1,opt2),end = "")
answer = int(input())
count = 0
while answer == opt1 + opt2:
    opt1 = random.randint(1,100)
    opt2 = random.randint(1,100)
    print("{:2d} + {:2d} = ".format(opt1,opt2),end = "")
    answer = int(input())
    count += 1

print("合计做对了{0}题".format(count))
```

运行结果：

```
41 + 52 = 93
76 + 19 = 95
26 + 61 = 87
64 + 12 = 76
36 + 67 = 103
44 + 47 = 91
75 + 62 = 137
82 + 62 = 144
28 + 38 = 66
77 + 100 = 177
34 + 23 = 57
18 + 17 = 35
1 + 97 = 98
84 + 55 = 133
合计做对了 13 题
```

在实际应用中，经常使用 while 循环确保用户的输入一定在一个合法的范围内，从而保证后续处理数据的可靠性。如下代码片段就可以确保用户的输入范围一定为 1～60。

```
num = int(input("请输入一个 1～60 的整数"))
while num < 1 or num > 60:
 num = int(input("请输入一个 1～60 的整数"))
print(num)
#执行到这里，一定是 1～60 的整数了
```

5.3 for 循环

利用 for 循环可以把序列中所有的元素按照原始的顺序遍历一遍，因为序列的大小是可知的，所以 for 循环可以看成是已知循环次数的循环。它的语法规则如下：

```
for 迭代变量 in 序列:
    语句块 1
else:
    语句块 2
```

其中，用到的序列通常是列表、元组、集合和 range 对象。语句块 1 是循环体，被执行的次数等同于序列的长度。和 while 循环类似，只有当 for 循环正常结束时才会执行语句块 2，else 部分同样可以省略。

Python 有一个内置的 range 迭代器类，它用于产生指定范围内的可迭代 range 对象。它经常被用于控制 for 循环的次数。迭代器类带来的一个优点是如果产生的元素数量巨大，它无须预先占用大量内存，而是每次迭代时才得到下一个元素。

它有两个调用形式，分别如下：

```
range(起始值, 终止值, 步长)
#产生起始值到终止值(不含)之间指定步长的迭代器对象
range(终止值)
#产生 0 到终止值(不含)之间的步长为 1 的迭代器对象
```

第一个调用形式的步长可以省略，如果省略就以 1 作为步长。

【例 5-3】 利用 for 循环和 range 对象编写程序计算 1～100(含 1 和 100)的所有整数的和。

利用 for 循环和 range 对象编写该程序，因为变量 i 的值是通过迭代器返回，所以几乎没有机会把它写成死循环。修改后的程序如下：

程序 5-3 求 1～100 的整数的和

```
1  #计算 1～100 的整数的和
2
3  res = 0
4  for i in range(1,101):
5      res += i
6  print("1 + 2 + 3 + … + 100 = ",res)
```

运行结果：

```
1 + 2 + 3 + … + 100 =  5050
```

【例 5-4】 小明是一个渔夫，坚持“三天打鱼，两天晒网”，已知每年的 1 月 1 日，小明是第一天打鱼，编写程序让用户输入一个年月日，输出当天小明是打鱼还是晒网。

根据题意知道小明的工作是 5 天一个周期，因此只要算出指定的日期是该年的第多少

天，然后用这个天数对 5 求余，如果余数是 1、2 或 3 就是在打鱼，否则就是晒网。

那么如何得到一个日期是该年的第多少天？显然，只要把该日期之前全部月份的天数累加，再加上这个日期的日就可以得到。众所周知，每月的天数除了 2 月之外的天数都是确定的，可以用一个包含 12 个整型元素的列表存储每月的天数。其中 2 月初始化为 28，然后根据用户的输入进行处理，如果是闰年则修改 2 月的天数为 29。有了每月天数的列表，那么后续的工作就是顺理成章的事情了。具体程序如下：

程序 5-4 三天打鱼两天晒网

```
#三天打鱼两天晒网

year,month,day = eval(input("请输入年月日(用逗号分开):"))
monthDays = [31,28,31,30,31,30,31,31,30,31,30,31]
#存储每月的天数
if year <= 0:
    print("年不合法")
elif month > 12 or month < 1:
    print("月不合法")
elif day < 1 or day > monthDays[month - 1]:
    print("日不合法")
else:
    if (year % 4 == 0 and year % 100!= 0) or year % 400 == 0:
        monthDays[1] = 29
    count = 0
    for i in range(month - 1):
        count += monthDays[i]
    count += day

    if count % 5 in [1,2,3]:
        print("小明在打鱼")
    else:
        print("小明在晒网")
```

运行结果：

```
请输入年月日(用逗号分开):2000,8,7
小明在晒网
```

5.4 break、continue 和 pass

5.4.1 break

前面讲到 while 和 for 循环在正常循环结束时，会执行 else 中的语句，言下之意就是循环可能非正常结束。循环的非正常结束主要包括程序出现异常退出和主动跳出两种情况。出现异常退出，这里先不讨论。所谓主动跳出，就是程序中有代码提前结束循环。

Python 提供了关键字 break 用于从循环中主动跳出，循环体中执行到 break 语句时立刻结束循环，程序执行流程转向循环之后的第一条语句。在 while 和 for 循环中都可以使用

break 语句。

【例 5-5】 传说爱因斯坦出了一个爬楼梯的数学题，有一个长长的楼梯，如果每次爬 2 阶，最后余 1 阶；如果每次爬 3 阶，最后余 2 阶；如果每次爬 5 阶，最后余 4 阶；如果每次爬 6 阶，最后余 5 阶；如果每次爬 7 阶，最后刚好爬完。请问这个楼梯至少多少阶？

基于计算思维来解决本题比较容易想到的方法是枚举测试，也就是从 1 开始逐步向上试验每个整数，测试每一个整数对 2、3、5、6 和 7 分别求余数的结果是否合乎题意。因为是从小到大逐步增加，所以找到的答案一定是最小的解。不做深入数学分析很难确定本题到底循环多少次，因此这可以看成未知循环次数的问题，那么适合用 while 循环解决。一个简单的实现程序如下。

程序 5-5-1 爱因斯坦楼梯

```
1  #爱因斯坦楼梯
2
3  steps = 0
4  while True:
5      steps += 1
6      if steps % 2 == 1 and steps % 3 == 2 and steps % 5 == 4
7  and steps % 6 == 5 and steps % 7 == 0:
8          print("此楼梯至少长度为",steps)
9          break
```

运行结果：

```
此楼梯至少长度为 119
```

上面的程序中有两个地方需要注意：第一，while 后面的条件写的是常量 True，也就是表示循环条件永远成立；第二，在找到并输出正确的解之后，就无须再循环，因此使用了 break 从循环中跳出，可见该循环并不是死循环。

现在的程序中每执行一次循环就把变量 steps 的值加 1，而根据题意可知正确的解一定是 7 的倍数，因此 steps 可以每次加 7 从而减少循环的次数。另外，根据题目的第一个条件可知正确的解一定是奇数，读者可自行考虑如何进一步减少循环的次数。

5.4.2 continue

关键字 continue 也是用于改变程序流程，它用于结束本轮循环。在 while 循环中遇到 continue 时直接跳转到循环条件判断；在 for 循环中遇到 continue 时直接取得循环下一个迭代的值。

例 5-5 的程序假设使用 continue，修改后的程序如下。

程序 5-5-2 爱因斯坦楼梯

```
1  #爱因斯坦楼梯
2
3  steps = 0
4  while True:
5      steps += 1
6      if steps % 2!= 1: continue          #不满足条件 1
```

```
7        if steps % 3!= 2: continue          # 不满足条件 2
8        if steps % 5!= 4: continue          # 不满足条件 3
9        if steps % 6!= 5: continue          # 不满足条件 4
10       if steps % 7!= 0: continue          # 不满足条件 5
11       print("此楼梯至少长度为",steps)
12       break
```

运行结果：

```
此楼梯至少长度为 119
```

这种写法相当于用的是排除法，一个整数只要不满足其中一个条件，就直接开始下一轮循环，换下一个整数来试验。

通常可以通过反写前面 if 的条件来避免使用 continue，两种写法的程序在效率上并无大的区别。但是实际应用中，把一些特殊情况列出来用 continue 处理，代码会具有良好的可读性。

5.4.3 pass

Python 中的 pass 是空语句，它不做任何事情，一般用作占位语句，主要是为了保持程序结构的完整性。

什么是保持程序结构的完整性呢？例如，有一个 for 循环，目前还不需要写任何循环体，此时就可以在内部写一个 pass 语句。这样这个循环就符合语法了，它也确实需要循环指定的次数，只是它的循环体不做任何事情，这样的循环属于空循环。

空循环有时被用于延时，如下代码片段可以输出 26 个大写字母，且每个字母输出之后会有个短暂的停顿。这个停顿是因为 j 需要在 range 返回的迭代器中遍历 30 万次，显然这个 30 万次所消耗的时间是与计算机硬件相关的，因此空循环延时难以精确控制消耗的时间。

```
>>> for i in range(26):
    print(chr(65 + i),end = "")
    for j in range(300000):
        pass
```

5.5 循环嵌套

如果在一个循环体内包含另外一个循环，那么就称为循环嵌套。其中 while 和 for 循环不仅可以自己嵌套，也允许互相嵌套。理论上循环嵌套的层次可以是任意层，但通常嵌套不要超过三层，否则代码可读性和可修改性都较差。

【例 5-6】 编写程序找出 10～1000 的所有完全数。所谓的完全数又称完美数或完备数。完全数是一种特殊的自然数，它的真因子的和等于它自己。

这个题目的基本解决办法是用循环把 10～1000 的整数全部遍历一遍，遍历时测试每一个整数是否是完全数。

程序 5-6 找 10～1000 的完全数

```
#找 10～1000 的完全数

for i in range(10,1001):                    #遍历求解空间
    temp = 0                                #存储因子的和,初始化为 0
    for j in range(1,i):                    #从 1 循环到 i - 1
        if i % j == 0:                      #判断是否为 i 的因子
            temp += j
    if temp == i:                           #判断 i 与真因子的和是否相等
        print(i)
```

运行结果：

```
28
496
```

【例 5-7】 编写程序解决《孙子算经》中的百钱百鸡问题。该问题描述如下：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

如果用数学方程组来解决该问题。设公鸡是 x 只，母鸡是 y 只，小鸡是 z 只，那么可以得到方程组如下：

$$\begin{cases} x + y + z = 100 \\ 5x + 3y + z/3 = 100 \end{cases}$$

但是该方程组只有两个方程却包含三个未知数，因此不是特别好解。此时可以采用枚举的计算思维进行验证，它的基本思想是根据题目的条件确定答案的范围，并在此范围内对所有可能的情况逐一验证，直到全部情况验证完毕。如果某个情况验证后符合题目的全部条件，则为本问题的一个解；若全部情况验证后都不符合题目的全部条件，则本题无解。

针对本例而言每种鸡的数量不可能小于 0，不可能大于 100，这就是大致的范围，具体的条件是百钱和百鸡。

因此可以得到表 5-2，其中第一列是公鸡数量，第二列是母鸡数量，第三列是小鸡数量，第四列表示百钱条件是否成立，第五列表示百鸡条件是否成立。

从表 5-2 可以看到，第一行数据三者都是 0，百钱和百鸡两个条件都不成立；当公鸡数量为 4、母鸡为 18、小鸡为 78 时，两个条件都成立，因此这就是一组满足条件的解。

表 5-2 百钱百鸡的枚举情况列表

公鸡数量/只	母鸡数量/只	小鸡数量/只	是否刚好百钱	是否刚好百鸡
0	0	0	False	False
0	0	1	False	False
0	0	2	False	False
…				
4	18	78	True	True
…				
4	19	77	True	False
…				
100	100	100	False	False

基于以上思想，可以用一个三层循环嵌套来实现验证过程。百钱百鸡问题枚举时有两个验证条件：一个是否刚好百钱；另一个是否刚好百鸡。但是实质上还有一个隐藏条件，小鸡数量一定是 3 的倍数。因此程序中需要一个保证小鸡数量对 3 求余是否等于 0 的判断。可以得到如下程序。

程序 5-7-1 百钱百鸡问题

```
1  #百钱百鸡问题
2
3  for cock in range(0,101):
4      for hen in range(0,101):
5          for chick in range(0,101):
6              if cock * 5 + hen * 3 + chick//3 == 100 and
7  cock + hen + chick == 100 and chick % 3 == 0:
8                  print("公鸡{0:2d} 母鸡{1:2d} 小鸡
9  {2:2d}".format(cock,hen,chick))
```

运行结果：

```
公鸡 0 母鸡 25 小鸡 75
公鸡 4 母鸡 18 小鸡 78
公鸡 8 母鸡 11 小鸡 81
公鸡 12 母鸡 4 小鸡 84
```

5.6 循环优化

例 5-7 的程序在执行时，使用了三层循环，其中的 if 语句会被执行的次数为 101 * 101 * 101＝1 030 301，大约 100 万次，那么这个程序是否可以优化呢？

现在再来仔细考虑一下该问题，因为每个公鸡是 5 钱，所以百钱最多只能买 20 只公鸡；同样每个母鸡是 3 钱，百钱最多只能买 33 只母鸡；另外，既然小鸡的数量一定是 3 的倍数。那么可以直接让小鸡的数量每次加 3。因此前面表格 5-2 就可以优化成表 5-3，其中的行数显著减少。

表 5-3 百钱百鸡优化后的枚举情况列表

公鸡数量/只	母鸡数量/只	小鸡数量/只	是否刚好百钱	是否刚好百鸡
0	0	0	False	False
0	0	3	False	False
0	0	6	False	False
…				
4	18	78	True	True
…				
20	33	99	False	False

根据表 5-3 很容易写出如下实现程序。

程序 5-7-2　百钱百鸡问题

```
#百钱百鸡问题优化 1

for cock in range(0,21):
    for hen in range(0,34):
        for chick in range(0,101,3):
            if cock * 5 + hen * 3 + chick//3 == 100 and cock + hen + chick == 100:
                print("公鸡{0:2d} 母鸡{1:2d} 小鸡
{2:2d}".format(cock,hen,chick))
```

和之前的代码相比，首先缩小了外面两层循环控制 range 函数的区间。其次在枚举小鸡数量时使用了 range 函数的第三个参数。因为 chick 一定是 3 的倍数，那么 if 条件中的判断 chick 是否是 3 的倍数的条件被删除了。

在这样优化后，在保证正确性的前提下程序比刚才简洁，此外该程序循环的次数已经变成了 21 * 34 * 34＝24 276。从 100 万次变成 2 万多次，降低了两个数量级，显然是一个巨大的进步。那么该问题是否可以进一步优化呢？答案是肯定的。

如果在枚举公鸡和母鸡的数量时，小鸡的数量通过 100 减去公鸡和母鸡的数量得到，就可以避免验证很多数量不是百鸡的情况。那么验证的过程就从前面的一个 5 列表格转换成了如表 5-4 所示的 4 列表格，其中小鸡的数量是利用 100 减去公鸡和母鸡的数量计算而来，因此只需要判断是否刚好是百钱。

表 5-4　百钱百鸡只枚举公鸡和母鸡的情况列表

公鸡数量/只	母鸡数量/只	小鸡数量/只	是否刚好百钱
0	0	100	False
0	1	99	False
0	2	98	False
…			
4	18	78	True
…			
20	33	47	False

根据表 5-4 的枚举方法实现的程序如下。

程序 5-7-3　百钱百鸡问题

```
#百钱百鸡问题优化 2

for cock in range(0,21):
    for hen in range(0,34):
        chick  = 100 - cock - hen
            if cock * 5 + hen * 3 + chick//3 == 100 and chick % 3 == 0:
                print("公鸡{0:2d} 母鸡{1:2d} 小鸡
{2:2d}".format(cock,hen,chick))
```

此时该程序已经减少了一层循环，中间的循环体判断是否满足百钱的条件，以及小鸡的

数量是否是 3 的倍数，也就是说程序已经从一个三层循环嵌套优化成了两层循环嵌套，此时总循环的次数是 21×34＝714。

现在已经从最初的循环大约 100 万次优化到了大约 700 次，降低了 4 个数量级。那么该问题是否可以进一步优化呢？答案是肯定的，再仔细看一下刚才的方程组：

$$\begin{cases} x + y + z = 100 & ① \\ 5x + 3y + z/3 = 100 & ② \end{cases}$$

将②乘以 3 再减去①，可以得到

$$14x + 8y = 200$$

一个方程两个未知数，计算思维的枚举法可以再次发挥作用。可以得到如下程序。

程序 5-7-4 百钱百鸡问题

```
#百钱百鸡问题优化 3

for cock in range(0,21):
    hen = (200 - cock * 14)//8
    if hen < 0: continue
    chick = 100 - cock - hen
    if cock * 5 + hen * 3 + chick//3 == 100 and chick % 3 == 0 :
        print("公鸡{0:2d} 母鸡{1:2d} 小鸡{2:2d}".format(cock,hen,chick))
```

从上面的程序可以看到，现在该问题已经变成了一层循环，合计循环 21 次，母鸡的数量通过表达式直接计算出来，然后再利用总数 100 的约束条件得到小鸡的数量，最后用百钱的条件判断是否是满足条件的解。

5.7 典型例题

【例 5-8】 编写程序让用户输入 x 和 n 的值，计算多项式的值，多项式的形式如下：

$$x^n + x^{n-1} + \cdots + x^2 + x + 1$$

输出结果时保留小数点后 2 位。

这样的题目实现时主要是找到合适的通项，然后就可以用循环解决。将上述多项式逆序并稍做整理后可以得到如下形式：

$$1 + x^1 + x^2 + \cdots + x^{n-1} + x^n$$

基于上述形式可以写出如下程序。

程序 5-8 计算多项式的值

```
#计算多项式的值

x = float(input("请输入浮点数 x:"))
n = int(input("请输入整数 n:"))

result = 1
item = 1
```

```
for i in range(n):
    item *= x
    result += item
print("{:.2f}".format(result))
```

运行结果：

```
请输入浮点数 x:3.2
请输入整数 n:7
49 97.33
```

【例 5-9】 有一个列表 lst 存储了 n 个 1～100 的正整数，编写程序统计出其中给的好数对的个数。好数对的描述如下：如果一组整数(i,j)满足 nums[i]==nums[j]并且 i <j，(i,j)就可以被称为是一组好数对。例如 lst=[1,4,5,1,1,5]，就有四组好数对，分别是(0,3)、(0,4)、(3,4)和(2,5)。

这个题目最容易想到的办法是枚举法，首先准备一个值为 0 的计数器变量，然后用两层循环实现列表的每个元素都和后面的元素逐个比较，遇到相等就可以认为找到一个好数对，因为和后面的元素比较的时候已经隐含了 i <j 这个条件。根据上述思想写出的代码如下。

程序 5-9-1 统计好数对的值

```
#统计好数对的值

lst = list(map(int,input("请输入多个用空格分开的 1～100 的整数").split()))
count = 0
for i in range(len(lst)):
    for j in range(i + 1,len(lst)):#j 从 i + 1 开始
        if lst[i] == lst[j]:
            count += 1
print("好数对的数量为:",count)
```

运行结果：

```
请输入多个用空格分开的 1～100 的整数 1 2 3 9 7 2 3 1
好数对的数量为: 3
```

下面换一个角度来考虑本问题。如果一个数字在列表中只出现了 1 次，那么该数字显然不会产生好数对；如果一个数字重复出现了 $n(n>1)$次，那么该数字出现好数对的数量其实是一个组合问题，就是从 n 个数字中每取出 2 个就可以产生一个好数对，根据组合数计算公式，该数字产生好数对 $n*(n-1)/2$ 个。这样的话，只需要统计出列表中每个数字产生的次数，然后对次数大于 1 的情况利用排列数进行计算并累加就可以得到最终结果。因为列表中的数字都为 1～100，所以可以另外用一个列表来存储每个数字出现的次数。基于上述思想可以写出如下代码，现在的代码多用了一个包含 100 个元素的列表，但是已经把代码变成了一层循环。

程序 5-9-2 统计好数对的值

```
#统计好数对的值

lst = list(map(int,input("请输入多个用空格分开的 1～100 的整数").split()))
temp = [0] * 100

for val in lst:#统计每个数字出现的次数
    temp[val - 1] += 1

count = 0
for val in [item for item in temp if item > 1]:
    #只需要处理出现次数大于 1 的情况
    count += val * (val - 1)//2

print("好数对的数量为:",count)
```

【例 5-10】 现有 A、B、C、D 四个犯罪嫌疑人被警察捉住，已经肯定其中有且只有一个人是罪犯。四人现在分别说了一句话，A 说他不是罪犯，B 说 C 是罪犯，C 说是 D 是罪犯，D 说 C 骗人。也已经知道其中有三个人说了真话，一个人说了假话，现在编写程序帮助警察找出罪犯。

本例如果依靠逻辑推理来解决，通常有排除法、推理法、列表法等。例如基于假设和列表可以给出表 5-5。

表 5-5 真值表

	如果是 A	如果是 B	如果是 C	如果是 D
A：不是我	False	True	True	True
B：是 C	False	False	True	False
C：是 D	False	False	False	True
D：C 骗人(不是 D)	True	True	True	False

表 5-5 的第一列是 A、B、C、D 的四句结论，第一行是假设 A、B、C、D 分别是罪犯的情况，然后对每一个单元格根据假设进行判断是否成立。例如：如果是 A，那么和 A 的结论“不是我”互相矛盾，可以在对应单元格写 False，同样 B 的结论“是 C”在此假设下也是不成立的，C 的结论“是 D”也不成立，只有 D 的结论“C 骗人”成立，因此这一列前三个单元格是 Fasle，最后一个单元格是 True。用此办法可以为每个单元格做出标记。

因为知道合计三个人说了真话，所以找到标记了三个 True 一个 Fasle 的那一列就是本例的解，就表 5.4 中数据可以看出 C 是罪犯。

就本例而言，可以把 A、B、C、D 放在列表中，然后用 for 循环对其进行枚举，进入循环后首先设置一个计数器变量 count＝0，然后用四条 if 语句对应四条结论，从而决定是否要给 count 加 1，每轮循环结束后检查 count 的值，如果为 3，说明成立了三条结论(也就是说三个人说了真话)，那么就找到了题目的解。基于上述思想可以编写出如下代码。

程序 5-10 谁是罪犯

```
#谁是罪犯

candidates = ['A','B','C','D']
for ch in candidates:
    count = 0
    if ch!= 'A': count += 1
    if ch == 'C': count += 1
    if ch == 'D': count += 1
    if ch!= 'D': count += 1
    if count == 3:
        print("罪犯是:",ch)
        break
```

运行结果：

```
罪犯是: C
```

【例 5-11】 约瑟夫环报数。$N(3 \leqslant N \leqslant 1000)$个人围成一圈，每人顺序编号，然后从 1 开始依次加 1 报数，当报数为 $k(1 \leqslant k \leqslant 9)$的倍数或者末尾是 k 时，删除此人(后面不再参加报数)。最后留下的是几号？

本例实质是一个稍作变化的约瑟夫环问题，可以编写程序模拟该过程，从而得到最终结果。借助于列表可以方便地解决本问题，首先将 $1 \sim N$ 的每个整数存储到列表中，然后对列表从左向右遍历，当遍历到最后一个元素时再回到第 0 个元素，遍历过程中遇到满足删除条件的元素时，将该元素删除，直到列表剩余一个元素为止。基于该思路可以得到如下代码。

程序 5-11-1 约瑟夫环报数

```
#约瑟夫环报数

n,k = list(map(int,input("请输入 n 和 k:").split()))
num = [0] * 1000
count = 1
step = 0
for i in range(n):                          #初始化数据
    num[i] = i + 1
while n > 1:
    if step == n:
        step = 0
        continue
    if count % k == 0 or count % 10 == k:
        for i in range(step,n - 1):         #删除
            num[i] = num[i + 1]
        n -= 1
    else:
        step += 1
    count += 1
print("最后留下{0}号".format(num[0]))
```

运行结果：

```
请输入 n 和 k:100 5
最后留下 47 号
```


当列表较大时列表删除是一个耗时操作，而刚才的解法中需要多次删除元素，为了提高效率，可以实现“假删除”，就本题而言可以把删除元素的值赋为－1，在计数时忽略值为－1的元素。此时可以得到如下程序。

程序 5-11-2 约瑟夫环报数

```
1  #约瑟夫环报数
2
3  n,k = list(map(int,input("请输入 n 和 k").split()))
4  num = [0] * 1000
5  count = 1
6  step = 0
7  length = n
8  while length > 1:
9      if step == n:
10         step = 0
11         continue
12     if num[step] == 1:
13         step += 1
14         continue
15     elif count % k == 0 or count % 10 == k:
16         num[step] = 1
17         length -= 1
18     count += 1
19     step += 1
20 for i in range(n):
21     if num[i] == 0:
22         print("最后留下{0}号".format(i + 1))
```

运行结果：

```
请输入 n 和 k:32 3
最后留下 19 号
```

习题

1. 从键盘输入一个十进制正整数，利用列表和除二取余法，计算出该数字的二进制值。
2. 从键盘输入一个十六进制正整数，计算出该数字的二进制值。
3. 随机生成一个 0～100 的整数，判断这个数是否等于 50，如果不等于则重新随机生成。最后输出一共随机生成了多少次。
4. 从键盘输入一个字母，如果输入的是小写英文字母，则将其转换为大写字母后显示输出；如果输入的是大写英文字母，则将其转换为小写字母后显示输出；如果既不是小写英文字母也不是大写英文字母，则原样显示。
5. 给定一个二进制字符串，例如"10100101"，计算并输出字符串中 0 的个数以及所有

数字之和。

6. 一副球拍售价 15 元，球 3 元，水 2 元。现在有 200 元，要求每种商品至少购买一个，有多少种可能正好把这 200 元花完？

7. 从键盘输入一批学生的成绩（成绩为整数），输入 0 或负数则输入结束，然后统计并输出优秀（大于或等于 90 分）、通过（60～59 分）和不及格（小于 60 分）的人数。

8. 用 * 输出一个等腰三角形。提示用户输入一个整数 n，代表输出的等边三角形由 n 行 * 组成。例如，输入 $n=3$。输出：

```
  *
 ***
*****
```

9. 用 * 输出一个正六边形，输入一个整数 n 代表输出的正六边形的边的长度（* 的数目）。

例如输入 $n=3$。输出：

```
  *   *   *
 *   *   *   *
*   *   *   *   *
 *   *   *   *
  *   *   *
```

10. 二维列表中有一组人员的姓名和年龄数据，编写程序找到这组数据中年龄最大的人，输出相关人员信息。

11. 计算 $s=a+aa+aaa+aaaa+\cdots+aa\cdots a$ 的结果，其中 a 是 0～9 的数字，有 n 个数相加（最大的数有 n 位）。例如 2＋22＋222＋2222＋22222（此时共有 5 个数相加）。编写程序，随机生成 a，从键盘输入 n，将公式进行输出，并输出计算结果。

12. 有一列表，存放有若干同学的姓名，编写程序将这些信息分成两组，元素顺序为偶数的放在一组，奇数的放在另一组，然后将分组的信息进行输出，输出格式如下：

```
分组        1      2      3      4
奇数组      张三   王五   孙七   吴九
偶数组      李四   赵六   周八
```

13. 假设某人每月计划给公交卡充一些钱用于坐公交，已知当地公交票费用按照表 5-6 进行计算。编写一个程序，从键盘输入充值金额，计算并输出充值的金额最多能坐多少次公交。

表 5-6　公交费用表

当月累计次数	票　　价
1～10	2 元（原价）
10～20	原价 9.5 折
21～50	原价 8 折
51 次以上	原价 5 折

14. 一条地铁线一共有 30 个站点，这些站点编号为 0～29，已知所有相邻站点间的距离

列表 distance，distance[i]表示编号为 i 的车站到编号为 i+1 车站间的距离，单位为 km，站距均为 1.5～3.0km。假设地铁的车票和距离有关，基础票价为 2 元(5km 内)，超过 5km 后每 5km 增加 1 元(不满 5km 的部分按 1 元计算)。编写一个程序，计算花 n 元最多能乘坐多少站。

15. 输出一个乘法表。要求输入一个整数 n，输出 $n*n$ 的乘法表，乘法表打印出来为下三角样式，格式工整。例如，输入 $n=4$。输出：

```
    1  2  3   4
1   1
2   2  4
3   3  6  9
4   4  8  12  16
```

15. 提示用户输入一个整型数 n(n 代表后续需要输入整型数的数量)，将 n 个整型数加起来并输出，如果输入的是非整型数则提示当前的输入非法，需要重新输入数值，如果输入 $n=0$ 则代表退出程序，否则继续提示用户输入新的 n。

例如：

```
Please input the number of numbers:(假设输入 n = 3)
Please input number 1: (假设输入 3)
Please input number 2: (假设输入 4)
Please input number 3: (假设输入 5)
```

输出：

```
sum = 12
Please input the number of numbers:
…
Please input the number of numbers:(假设输入 n = 0,则退出程序)
```

17. 提示用户输入一个整数 n，然后输出 [1,n)的所有的素数。例如，输入 $n=10$。输出：2,3,5,7。

18. 矩阵相加：提示用户输入一个数字 n，为矩阵的行数，再提示用户输入一个数字 m，为矩阵的列数，接下来，提示用户输入 $2*n*m$ 个数字(每次输入一个数字)。输出 C=A+B。

例如，输入：

```
Please input the number of rows:(假设输入 n = 2)
Please input the number of columns:(假设输入 m = 3)
Please input A[0,0]: 1
Please input A[0,1]: 1
Please input A[0,2]: 1
Please input A[1,0]: 1
Please input A[1,1]: 1
Please input A[1,2]: 1
```

```
Please input B[0,0]: 2
Please input B[0,1]: 2
Please input B[0,2]: 2
Please input B[1,0]: 2
Please input B[1,1]: 2
Please input B[1,2]: 2
```

输出：

```
C = [[3, 3, 3], [3, 3, 3]]
```

19. 有一个弹性小球从 H（$H \geqslant 100$）m 的高度做自由落体运动，每次落地后会反弹，反弹高度为上次下落高度的一半，然后继续下落，如此反复（假设反弹和下落不会停止）；编写程序，计算并输出小球第 N 次落地时，共经过了多少米，第 N 次反弹的高度为多少。