第5章 循环结构

学习目标：

- 掌握 while 语句的语法规则、执行过程和使用方法。
- 掌握 for 语句的语法规则、执行过程和使用方法。
- 理解循环结构的嵌套。
- 掌握循环的中途退出的表示方法。
- 掌握循环结构程序设计方法及典型算法。

在前面的章节中，学习了 Python 语言程序设计的顺序结构和选择结构。然而在进行程序设计时，经常需要实现当某一条件成立时，重复一些操作的功能，实现这一功能就需要用到循环结构。

在 Python 语言中，一般用 while 语句和 for 语句实现循环结构。

5.1 while 循环和哨兵循环

当某段代码需要重复执行时，可以用循环结构实现。例如，要实现每隔 1s 在屏幕上打印一次“hello world”并持续打印一个小时，如果直接把 print('hello world')这句代码重复书写 3600 遍，将会大大增加代码量。相比之下，一个简单的循环结构便可以实现这个功能。本节主要介绍 while 循环和哨兵循环。

5.1.1 while 循环


微课视频

Python 中，while 循环和 if 条件分支语句类似，即在条件（表达式）为真的情况下，会执行相应的语句块。不同之处在于，只要条件为真，while 就会一直重复执行该语句块，一直到条件为假时才结束这一重复过程。while 循环的语法形式为

```
while 条件表达式 :
    循环体语句/语句块
```

while 语句执行的具体流程为：首先判断条件表达式（该条件表达式又称为循环条件）的值，值为真（True）时，则执行语句块（这部分语句块又称为循环体）。当执行完一次循环

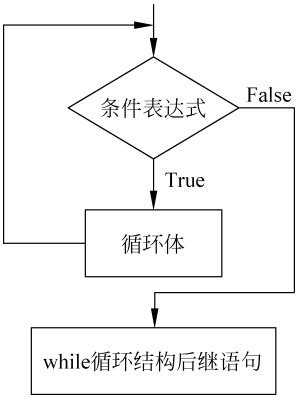

图 5-1 while 循环的执行流程

体后，再重新判断循环条件是否为真，若仍为真，则继续重复执行循环体。如此循环，直到循环条件为假(False)时，才结束循环，去执行该循环结构后面的语句。

while 循环结构的执行流程如图 5-1 所示。

【例 5-1】 利用 while 循环找出第一个大于 50 的 3 的乘方。

```
In : product = 3
     while product <= 50:
           product *= 3
     print(product)
Out: 81
```

在 while 语句中，为了避免死循环(也就是无限次重复执行循环体)的情况出现，在循环体中一定要存在改变循环条件的语句，或者有 break 语句。如在例 5-1 中，将 while 循环中的 product *= 3 代码注释掉，再次运行程序会发现，程序没有任何输出结果，这是因为循环永远不会结束(因为变量 product 的值一直为 3，没有发生改变，则循环条件 product <= 50 一直为 True)，除非强制结束该程序的运行。

位于 while 循环体中的语句必须使用相同的缩进格式(通常缩进 4 个空格)，以构成一个语句块，否则 Python 解释器会报 SyntaxError 错误(语法错误)。

除此之外，while 循环还常用来遍历列表、元组和字符串，因为它们都支持通过下标索引获取指定位置的元素。

【例 5-2】 while 循环遍历字符串变量。

```
In : my_char = "Hello World!"
     i = 0;
     while i < len(my_char):
         print(my_char[i],end = "")
         i = i + 1
Out: Hello World!
```

【例 5-3】 while 循环计算 n 的阶乘。

```
In : i = 1
     fac = 1
     n = eval(input("please input a integer:"))
     while (i <= n):
         fac *= i
         i = i + 1
     print("n = %d,fac = %d" % (n, fac))
Out: please input a integer: 5
     n = 5,fac = 120
```

在例 5-2 和例 5-3 中，循环体的执行次数是由变量 i 进行控制，在循环体中不可或缺的语句是改变变量 i 值的语句(例如，在这两个例子中的 i=i+1 语句)，像这种用来控制循环执行次数的变量 i，称为循环变量。

注意：

while 语句的使用规则如下。

（1）while 语句中的表达式一般是关系表达式或逻辑表达式，只要表达式的值为真（非 0）即可继续循环。

（2）注意在循环体中改变循环变量的值，以避免死循环。

5.1.2　哨兵循环

当循环的次数在编程时无法确定，而需要用户触发时，应该采用哨兵循环。在哨兵循环中，哨兵是一个特殊值，它被用来确定何时停止循环，即在循环过程中遇到设定的特殊数据（哨兵值），循环语句才会终止。但是，在采用哨兵循环结构编程时，也同样需要注意，在循环体中要更新哨兵。


微课视频

【例 5-4】 哨兵循环使用示例：要求输入若干学生成绩并计算平均值。

方法一（需要用户预先输入学生人数）：

```
In : stu_amount = int(input('请输入学生人数: '))
     sum = 0; i = 0
     while (i < stu_amount):
         i = i + 1
         print('请输入第',i,'名同学的成绩',end = '')
         grade = int(input(''))
         sum += grade
     if stu_amount <= 0:
        print('未输入成绩')
     else:
        aver = sum / stu_amount
     print(f'平均成绩为: {aver :.2f}')
```

程序的运行结果如下。

```
Out: 请输入学生人数: 5
     请输入第 1 名同学的成绩: 97
     请输入第 2 名同学的成绩: 82
     请输入第 3 名同学的成绩: 88
     请输入第 4 名同学的成绩: 75
     请输入第 5 名同学的成绩: 70
     平均成绩为: 82.40
```

方法二（设定哨兵为成绩中不可能出现的数值－1，来控制循环结束）：

```
In : sum = 0; counter = 0
     grade = int(input('请输入学生成绩(－1 结束输入): '))
     while grade != －1:
         sum += grade
         counter += 1
         grade = int(input('请输入学生成绩(－1 结束输入): '))
     if counter == 0:
        print('未输入成绩')
     else:
        aver = sum / counter
     print(f'平均成绩为: {aver :.2f}')
```

程序的运行结果如下。

```
Out: 请输入学生成绩(-1结束输入): 97
     请输入学生成绩(-1结束输入): 82
     请输入学生成绩(-1结束输入): 88
     请输入学生成绩(-1结束输入): 75
     请输入学生成绩(-1结束输入): 70
     请输入学生成绩(-1结束输入): -1
     平均成绩为: 82.40
```

5.2 for 语句和循环嵌套

Python 中的循环控制语句主要有两种，分别是 while 和 for，前面章节已经详细介绍了用 while 语句构造的循环结构，本节将介绍用 for 语句构造循环结构，以及循环嵌套结构。在下文中，将用 while 语句构造的循环结构称为 while 循环，用 for 语句构造的循环结构称为 for 循环。


微课视频

5.2.1 for 循环表达式及流程图

for 语句多用于遍历字符串、列表、元组、字典、集合等可迭代对象中的元素。for 语句的语法格式如下。

```
for 迭代变量 in 字符串|列表|元组|字典|集合(可迭代对象):
    循环体语句/语句块
```

for 语句将会自动从字符串、列表等可迭代对象中依次取出序列元素(实际上能自动依次读取的原因是可迭代对象中包含__iter__()方法，因此迭代变量将会依次取得序列中的元素，所以，在循环前不需要对迭代变量赋初值，在循环体中也不需要用语句改变迭代变量的值；循环体由具有相同缩进格式的多行语句构成(和 while 一样)。

for 循环结构的执行流程如图 5-2 所示。

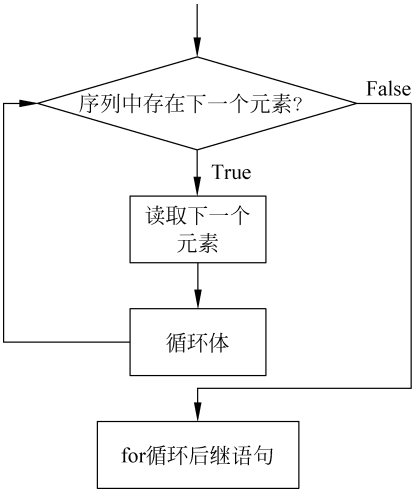

图 5-2 for 循环的执行流程

【例 5-5】　利用 for 循环判断用户输入的字符串中的字符，如果字符串中包含小写字母，则将其变成大写字母，输出字符串，要求字符之间由一个空格分隔。

```
In : for char in 'Python3.0':
        if 97 <= ord(char) <= 122:
           char = chr(ord(char) - 32)
           print(char, end = '')
        else:
           print(char, end = '')
Out: P Y T H O N 3 . 0
```

在 Python 中，for 循环也可以转换为 while 循环。其具体的转换过程如图 5-3 所示。

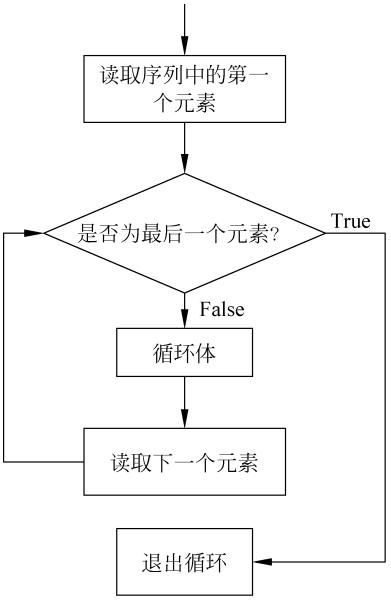

图 5-3　和 for 循环对应的 while 循环

5.2.2　for 循环结构综合举例

【例 5-6】　利用 for 循环统计用户输入的字符串中的字母、数字和其他字符的数量。


```
In : letter = 0; num = 0; other = 0
    s = input('请输入字符串: \n')
    for char in s:
        if char >= 'a' and char <= 'z' or char >= 'A' and char <= 'Z':
           letter += 1
        elif '9' >= char >= '0':
           num += 1
        else:
           other += 1
    print(f'字符串中共有{letter:2d}个字母,{num:2d}个数字,{other:2d}个其他/字符')
Out: 请输入字符串: ABc123++
    字符串中共有 3 个字母,3 个数字,2 个其他字符
```

【例 5-7】　利用 for 循环计算 Fibonacci 数列的前 20 项(1、1、2、3、5、8、…)，要求每行显

示 4 项。Fibonacci 数列的数学公式如下。

$$\begin{cases} F_1 = 1, & n = 1 \\ F_2 = 1, & n = 2 \\ F_n = F_{n-1} + F_{n-2}, & n \geqslant 3 \end{cases}$$

```
In : f1 = 1; f2 = 1
    for i in range(1, 11):
        print(str.format("{0:6}{1:6}", f1, f2), end = ",")
        if i % 2 == 0: print()
        f1 += f2; f2 += f1
```

程序的运行结果如下。

```
Out:      1     1,     2     3,
          5     8,    13    21,
         34    55,    89   144,
        233   377,   610   987,
       1597  2584,  4181  6765,
```

5.2.3 循环嵌套

Python 不仅支持 if 语句相互嵌套，同样也支持 while 和 for 语句相互嵌套。在一个循环体内又包含另一个完整的循环结构，则称为循环的嵌套。在多层循环结构中，for 循环结构和 while 循环结构可以相互嵌套。多重循环的总循环次数等于每一重循环次数的乘积。

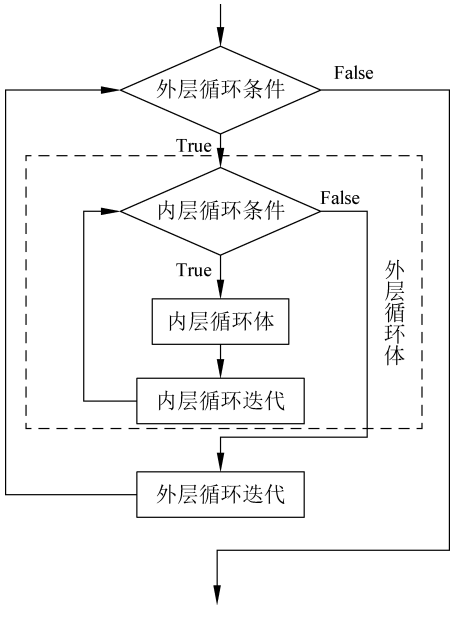

图 5-4 循环嵌套的执行流程图

当多个循环结构相互嵌套时，位于外层的循环结构称为外层循环或外循环，位于内层的循环结构称为内层循环或内循环。根据上面的分析，假设外层循环的循环次数为 n 次，内层循环的循环次数是 m 次，则内层循环的循环体实际上需要执行 n×m 次。循环嵌套的执行流程如图 5-4 所示。

当遇到循环嵌套时，Python 解释器的执行流程如下。

(1) 当外层循环条件为 True 时，则执行外层循环结构中的循环体。

(2) 外层循环体中包含语句和内层循环。当内层循环的循环条件为 True 时，则执行内层循环中的循环体，直到内层循环条件为 False，跳出内层循环。

(3) 如果此时外层循环的条件仍为 True，则返回第(2)步，重新开始一次新的外层循环，继续执行外层循环体，直到外层循环的循环条件为 False。

(4) 当外层循环的循环条件为 False 时，整个循环嵌套执行完毕。

【例 5-8】　while-for 嵌套结构示例。

```
In : i = 0
    while i < 3:
        for j in range(0,2):
            print("i = ",i,"j = ",j)
        i = i + 1
Out: i = 0  j = 0
    i = 0  j = 1
    i = 1  j = 0
    i = 1  j = 1
    i = 2  j = 0
    i = 2  j = 1
```

例 5-8 程序中运用了循环嵌套结构，其中，外层循环使用的是 while 语句，内层循环是 for 语句。当进入循环嵌套时，循环变量 i 初始值为 0，这时即进入外层循环。当进入外层循环后，内层循环把 i 当成一个普通变量，值为 0，开始执行内层循环，直到 j=1 不满足循环条件，跳出 for 循环体，继续执行 while 外层循环的循环体；执行 i = i+1 语句，如果 i<3 依旧成立，则重复执行上述步骤。直到 i<3 不成立，此循环嵌套结构才执行完毕。

此程序中外层循环将循环 3 次(从 i=0 到 i=2)，而每次执行外层循环时，内层循环都从 j=0 循环执行到 j=1。因此，该内层循环体执行了 3×2=6 次。

5.3　可迭代对象

Python 中可迭代对象(iterable)并不是指某种具体的数据类型，而是指存储了元素的一个容器对象，且容器中的元素可以通过__iter__()方法或__getitem__()方法访问。__iter__()方法的作用是让对象可以用 for … in 的形式循环遍历，__getitem__()方法是让对象可以通过“实例名[index]”的方式访问实例中的元素。

迭代是访问集合元素的一种方式。迭代器是一个可以记住遍历位置的对象。迭代器对象从集合的第一个元素开始访问，直到所有元素被访问完结束。生成器和迭代器的功能非常相似，它提供__next__()方法，这意味着程序可以调用内置的 next()函数来获取生成器的下一个值，也可以使用 for 循环来遍历生成器。

生成器与迭代器的区别在于：迭代器通常是先定义一个迭代器类，然后通过创建实例来创建迭代器；而生成器则是先定义一个包含 yield 语句的函数，然后通过调用该函数来创建生成器。

常见的可迭代对象包括。

(1) 集合数据类型，如 list、tuple、dict、set、str 等。

(2) 生成器(generator)，包括生成器和带 yield 的生成器函数(generator function)。

【例 5-9】　可迭代对象示例。

```
In : sum = 0
    for n in [2, - 3,0,17,9]:
        sum += n
    print(sum)
Out: 25
```

5.4 内置函数 range

Python 中，内置函数 range() 经常被应用于生成可迭代对象。在程序开发时，它的身影无处不在，通常用来控制循环，控制数值范围等。


微课视频

5.4.1 range 函数使用规则

Python 中的内置函数 range()可以创建一个整数列表，一般用在 for 循环中。其函数语法格式如下。

```
range(start, stop[, step])
```

其中，各参数说明如下。

(1) start：计数从 start 开始。默认是从 0 开始。例如，range(5)等价于 range(0,5)。

(2) stop：计数到 stop 结束，但不包括 stop。例如，range(0,5)是[0,1,2,3,4]，不包含 5。

(3) step：步长，默认为 1。例如，range(0,5)等价于 range(0,5,1)。

在 Python 2 中 range 的类型为函数，是一个生成器；Python 3 中 range 的类型为类，是一个迭代器，此时 range()返回的是一个可迭代对象(类型是对象)，而不是列表类型。

【例 5-10】 创建一个从 0 到 9 的数字序列，并进行打印输出。

```
In : for counter in range(10):
        print(counter, end = '')
Out: 0123456789
```

range()函数的参数可以是一个、两个或者三个。不同的参数有不同的定义和用法。

(1) 一个参数：产生从 0 到参数值的连续整数序列，但不包括参数的值。索引 0 作为默认参数存在。例如，range(5)等价于 range(0,5)，实际产生 0～4 的整数序列。

(2) 两个参数：产生从第一个参数值到第二个参数值的整数序列，不包括第二个参数值。例如，range(4,6)可以产生数字序列[4,5]。

(3) 三个参数：从第一个参数值开始索引，按照指定步长递增，直至第二个参数值，且不包括第二个参数值。例如，range(4,8,2)可以产生数字序列[4,6]。

【例 5-11】 range()函数使用示例：利用 for 循环计算 1～100 中的所有偶数和以及奇数和。

```
In : odd_sum = even_sum = 0
    for i in range(1,100,2):
        odd_sum += i
    for i in range(0,101,2):
        even_sum += i
    print('偶数和为: %.2f, 奇数和为: %.2f'%(even_sum,odd_sum))
Out: 偶数和为: 2550.00, 奇数和为: 2500.00
```

5.4.2 off-by-one 错误

大小差一(off-by-one)错误主要指程序从缓冲区读入数据时发生溢出，并且只是溢出缓

冲区一个字节。这种错误的产生通常与边界验证不严和字符串操作有关。例如：

```
In : list = ['a','b','c','d']
     for i in range(5):
         print(list[i])
```

上述代码定义了长度为 4 的列表 list，利用 for 循环遍历列表中的所有元素，并进行打印输出。循环下标从 0 开始到 4，出现了 list[4]这样不存在的列表元素。这就是典型的“大小差一错误”。

5.5　break 语句、continue 语句和 else 子句

在前面的几个章节中，学习了 Python 中的两大循环控制语句。无论是 for 循环还是 while 循环，只有在循环条件变为 False 时，循环才会终止。如果想要控制循环的执行，例如，在例 5-2 中循环遍历“Hello World”字符串时，只打印输出“Hello”字符串；在例 5-10 中创建 0～9 的数字序列时，只打印奇数。这些流程的控制均可以借助本节介绍的 break、continue 和 else 语句来实现。

5.5.1　break 语句

break 语句用在 while 和 for 循环中，用来终止循环语句，即可以在循环条件未变为 False 或者 for 语句的迭代对象还没被完全迭代时跳出循环结构，停止执行循环体。

【例 5-12】 break 语句使用示例。

```
In : for number in range(1, 10):
        if number % 3 == 0:
           break
        print(number, end = ' ')
Out: 1 2
```

分析程序可以看出，当循环至 number＝3 时，条件表达式 number%3＝＝0 为 True，执行 break 语句，直接终止当前的循环，跳出循环体。break 语句一般会结合 if 语句进行搭配使用，表示在某种条件成立时跳出循环体。

另外，在循环嵌套结构中，break 语句只会终止所在循环体的执行，跳出当前的循环结构，而不能跳出所有的循环体。为了用 break 语句跳出循环嵌套的外层循环，可先定义 bool 类型的变量来标志是否需要跳出外层循环，然后在内、外层循环中分别使用两条 break 语句来实现。

【例 5-13】 循环嵌套中的 break 使用示例。

```
In : flag = False
     for i in range(7):              #外层循环
         for j in range(5):          #内层循环
             if j == 3:
                flag = True
                break
```

```
            print(j,end = "")
        print("\nj = ",j,"时跳出内层循环")
        if flag == True:
            print("j = ",j,"时跳出外层循环")
            break
```

上面程序中，提前定义了一个 bool 类型的变量 flag，并为其赋初值 False。在内层循环中判断 j 是否等于 3，当 j 等于 3 时，程序将 flag 设为 True，并跳出内层循环；接下来程序继续执行外层循环的剩余语句，由于 flag 为 True，因此执行外层的 break 语句来跳出外层循环。程序的运行结果如下。

```
Out: 012
    j = 3 时跳出内层循环
    j = 3 时跳出外层循环
```


微课视频

注意：

break 语句的使用规则如下。

(1) break 语句只能用于循环结构中。

(2) break 语句在循环体中，一般与 if 语句配合使用。

(3) 在多层循环中，一个 break 语句只向外跳一层。如果需要跳转到最外层，则需要使用多个 break。

5.5.2 continue 语句

continue 语句和 break 语句类似，也是用在 while 和 for 循环中。区别是 continue 只是忽略当次循环中 continue 后面剩下的语句，直接开始是否进行下一次循环的判断，并不会跳出循环体，终止循环；而 break 则是完全终止循环。和 break 一样，在循环嵌套结构中，continue 语句只作用于当前层的循环结构。

【例 5-14】 continue 使用示例。

```
In : for letter in 'Python':
        if letter == 'h':
           continue
        print ('当前字母 :', letter)
Out: 当前字母 : P
    当前字母 : y
    当前字母 : t
    当前字母 : o
    当前字母 : n
```

从上面的运行结果可以看出，当 letter 等于 h 时，程序没有输出“当前字母：h”字符串，因为程序执行到 continue 时，忽略了当次循环中 continue 语句后面的代码。由此可以看出，如果把一条 continue 语句放在当次循环的最后一行，那么这条 continue 语句是没有任何意义的。因为它仅忽略了一片空白，没有忽略任何程序语句。

【例 5-15】 输出 200～400 中 5 的倍数，要求每行显示 8 个数。

```
In : i = 0
    for item in range(200, 401):
        if item % 5 != 0:
            continue
        print(item,end = " ")
        i = i + 1
    if(i % 8 == 0):print()
```

程序的运行结果如下。

```
Out: 200 205 210 215 220 225 230 235
    240 245 250 255 260 265 270 275
    280 285 290 295 300 305 310 315
    320 325 330 335 340 345 350 355
    360 365 370 375 380 385 390 395
    400
```


微课视频

5.5.3　else 子句

Python 中，无论是 while 循环还是 for 循环，后面都可以紧跟着一个 else 代码块，它的作用是当 while 循环条件为 False，或者 for 语句迭代对象没有下一个元素时，程序会跳出循环，执行 else 代码块。需要注意的是，当用 break 语句结束循环，跳出循环体时，else 子句也会被跳过，不会被执行。也就是说，else 子句只有在 while 循环条件不成立或者 for 循环迭代对象迭代完成后，才会被执行。语法格式如下。

```
for 变量 in 可迭代对象:
  循环体语句(块)1
else:
  语句(块)2
```

或者：

```
while 条件表达式 :
    循环体语句(块)1
else:
  语句(块)2
```

【例 5-16】 使用 for 循环的 else 子句。

```
In : for n in range(2, 6):
        for x in range(2, n):
            if n % x == 0:
               print(n, '等于', x, '* ', n//x)
               break
        else:
            print(n, '是质数')
Out: 2 是质数
    3 是质数
    4 等于 2 * 2
    5 是质数
```

例 5-16 中，在外层循环中 n 分别取值为 2,3,4,5，在内层循环中 x 分别取值为 2～n，在内层循环体内判断 n 是否能被 x 整除。当 n=2 时，内层循环迭代对象为空，这个时候执行 else 子句，打印“2 是质数”；当 n=3 时，进入内层 for 循环体，但是不符合 n % x == 0(3%2==0)的条件，不执行 break 语句，接着便会执行 else 子句，打印“3 是质数”；当 n=4 时，进入内层 for 循环体，此时满足 n % x == 0(4%2==0)条件，打印“4 等于 2 * 2”，之后执行 break 语句，此时已经跳出内层循环，不执行 else 子句；当 n=5 时，进入内层 for 循环体，每次循环都不符合 n % x == 0 的条件，在内层 for 循环迭代结束后，执行 else 子句，打印“5 是质数”。至此整个程序执行完毕。

5.5.4 标志变量

flag 是 Python 程序中常用的一个标志变量(布尔类型变量)。flag 在程序中作为一个标识，用于控制执行流程和进行逻辑判断，通常使用布尔类型变量中的 False 和 True 来表示。


微课视频

【例 5-17】 请画出判断一个整数 m 是否为质数的流程图(图 5-5)。

判断变量 m 是否为质数时，首先读入数据并赋值给变量 m，初始化循环变量 i=2，接着进入一个循环结构，i 从 2 循环到 m－1，循环条件是判断是否 i<m，若为 False，说明所有的 i 都已经过测试，此时退出循环；若为 True，则测试 i 是否能被 m 整除：如果整除的结果 r==0，说明可以被整除，则执行 break 语句，退出循环；否则令循环变量加 1，继续循环。从流程图(图 5-5)中可以看出，退出该循环结构的出口有两条：①号出口代表不满足循环条件的出口，表示变量 i 从 2 循环到 m－1 均不能被 m 整除，所以，从这个出口退出循环意味着 m 是质数，此时 i >= m；②号出口代表通过 break 语句退出循环的出口，表示存在能被 m 整除

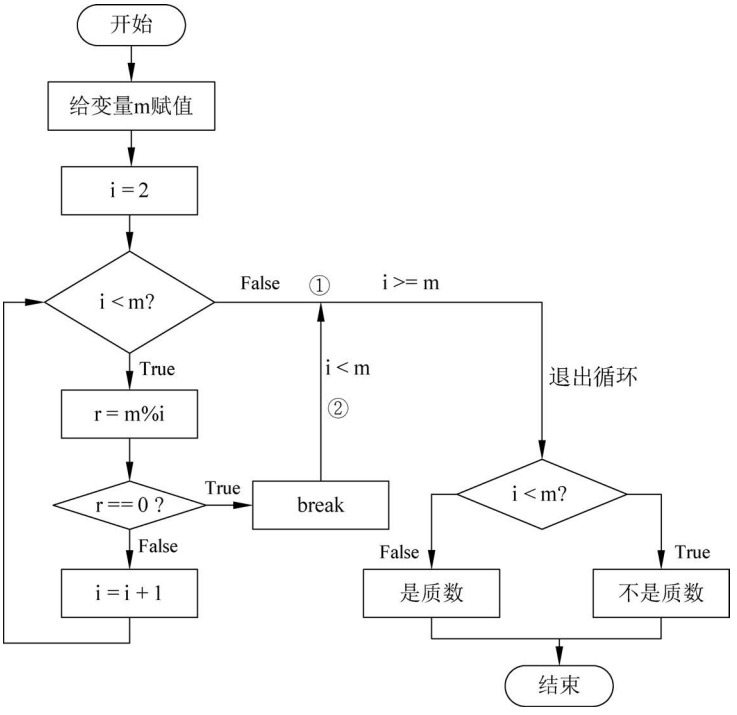

图 5-5 质数判断流程图

的变量 i，所以从此出口退出循环意味着 m 不是质数，此时 i＜m。在循环结构后通过一个双分支结构来判断退出循环结构的出口类型，从而得出 m 是否是质数的结论。

通过判断循环结构出口类型来判断 m 是否为质数的方法并不直观，所以在编程过程中，通常会引入标志变量 flag 来实现判断功能，通常取值为布尔类型变量中的 True 和 False。在本例中，使用 True 代表是质数，False 代表不是质数。在使用前，需要初始化标志变量，假设 m 是质数，则 flag 初始化为 True。如果在循环过程中存在一个 i 能被 m 整除时，在执行 break 语句之前，需要将 flag 设置为 False 表明 m 不是质数。使用标志变量判断质数的完整流程如图 5-6 所示。

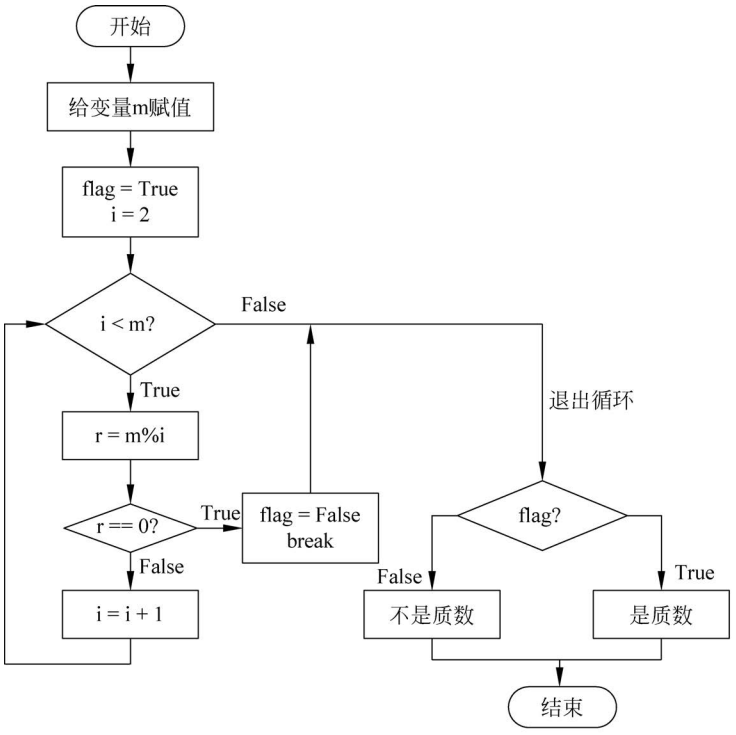

图 5-6　带标志变量的质数判断流程图

首先读入数据并赋值给变量 m，初始化标志变量 flag＝True，循环变量 i＝2，接着进入循环结构，变量 i 从 2 循环到 m－1，在循环体中测试 i 是否能被 m 整除，并在执行 break 语句之前，设置 flag 的值为 False。在退出循环结构后添加一个双分支结构，根据 flag 不同的取值，判断 m 是否为质数。

5.6　初识数据科学

探索数据是数据科学项目的第一步。在数据科学中，经常会使用统计学来描述和总结数据。描述性统计分析是关于数据的描述和汇总，它主要使用以下两种方法。

(1) 定量方法以数值方式描述和汇总数据。

(2) 可视化方法通过图表、曲线图、直方图和其他图形来说明数据。具体的实现方法将在第 11 章中介绍。

本节主要介绍 Python 中常用的几种描述性统计信息的内置函数，它们的含义如下。

(1) min()：集合中的最小值。

(2) max()：集合中的最大值。

(3) range()：最小值到最大值的范围。

(4) count()：集合中值的数量。

(5) sum()：集合中所有值的总和。

【例 5-18】 用户随机输入某一课程中 5 名学生的成绩绩点，编写代码，计算 5 名学生绩点总和，并输出绩点的最大值和最小值。

```
In :
    score = []
    for i in range(5):
       s = float(input(请输入学生绩点:'))
       score.append(s)
    total = sum(score)
    maxScore = max(score)
    minScore = min(score)
    print('学生绩点总和:',total)
    print('绩点最大值:',maxScore,'绩点最小值:',minScore)
Out: 请输入学生绩点:2.5
    请输入学生绩点:3.8
    请输入学生绩点:3.5
    请输入学生绩点:3.4
    请输入学生绩点:2.9
    学生绩点总和:16.1
    绩点最大值:3.8 绩点最小值:2.5
```

5.7 实践与练习

【实践 1】 求 1!+2!+…+n!，其中，n(1≤n≤10)值由用户输入。

```
In : n = int(input('请输入一个 1 到 10 之间的正整数'))
    fac = 1
    sum = 0
    i = 1
    while n >= i:
       fac = fac * i
       sum = sum + fac
       i = i + 1
    print(sum)
```

【实践 2】 编写程序，判断一个数是否为质数。

方法一：

```
In : num = int(input('输入一个任意的大于 1 的整数:'))
    for i in range(2,num):
        r = num % i
        if (r == 0):
            print(num,'不是质数')
```

```
            break
    else:
         print(num,'是质数')
```

方法二：

```
In : num = int(input('输入一个任意的大于 1 的整数:'))
     i = 2
     flag = True
     while i < num:
         if num % i == 0:
             flag = False
               break
         i += 1
     if flag:
         print(num,'是质数')
     else:
         print(num,'不是质数')
```

【练习】 设计知识竞赛方案。如果用户连续三次回答错误，那么比赛应该停止并提示“答题失败”。如果比赛正常结束，则输出答题时间和答题准确率。要求竞赛至少包含 20 个与知识相关的话题。

设计思路：本题采用循环结构实现用户不断答题的功能，首先需要确定题目是否作答完毕，若所有题目作答完毕，则比赛结束，输出答题时间和准确率；否则，继续作答，同时需要判断答案对错。此时，需要设置一个变量 flag 来判断“是否连续三次回答错误”：每答错一次，flag＝flag＋1，当 flag＝＝3 时，结束比赛，提示“答题失败”；若中间穿插一次答对的情况，则 flag 重新置 0，用户可以继续作答，直至答题结束。具体的比赛设计流程如图 5-7 所示。

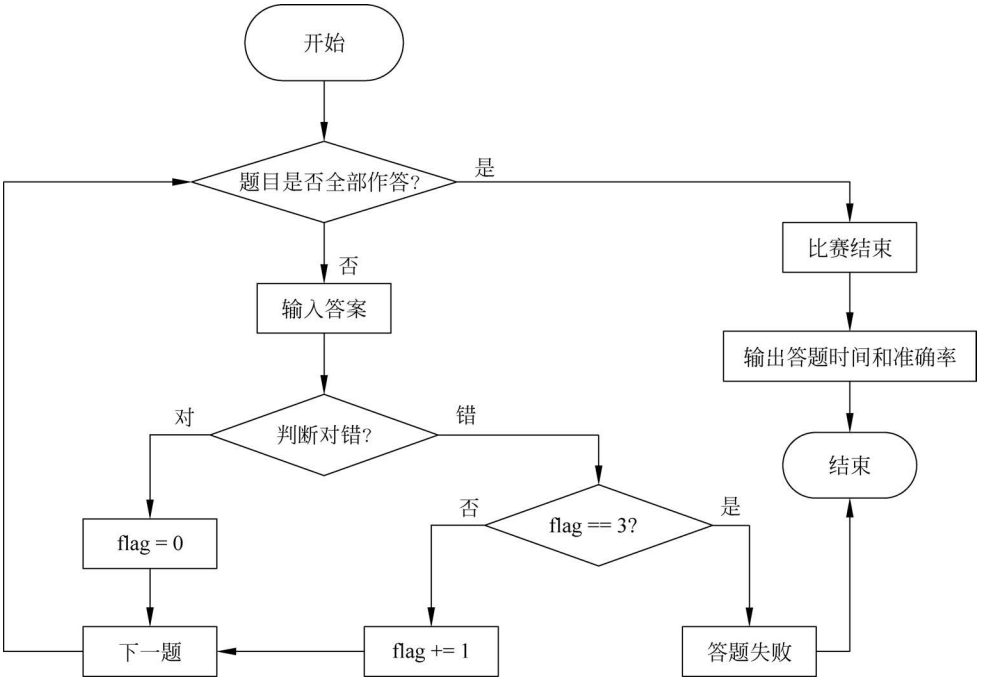

图 5-7　知识竞赛设计流程图

小结

1. 循环语句的基本形式

Python 语言提供了两种循环语句：while 语句和 for 语句。两者的基本形式如下。

(1) while 语句。

```
while(条件表达式):
    循环体语句/语句块
```

(2) for 语句。

```
for 迭代变量 in 字符串|列表|元组|字典|集合(可迭代对象):
   循环体语句/语句块
```

2. 循环语句的使用规则

(1) while 语句和 for 语句属于“当型”循环，即“先判断，后执行”。

(2) 建立循环常见以下几种情况。

① 循环次数未知的，即循环次数及控制条件要在循环过程中才能确定，此种情况适合用 while 语句来编程。

② 循环次数已知的，即在循环之前就能确定循环控制变量的初值、步长及循环次数，此种情况适合用 for 语句来编程。

③ 两种循环语句可以相互嵌套组成多重循环。循环之间可以并列但不能交叉。

④ 在循环程序中应避免出现死循环，即应保证循环变量的值在运行过程中可以得到修改，并使循环条件逐步变为假，从而结束循环。

⑤ 在循环体中出现的 break 语句和 continue 语句能改变循环的执行流程。它们的区别在于：break 语句将终止整个循环的执行，跳出循环结构；而 continue 语句只能结束本次循环，继续执行新一轮的循环。

(3) for、while 语句可以附带一个 else 子句，如果 for、while 语句没有被 break 语句中止，则会执行 else 子句，否则不执行。语法格式如下。

```
for 变量 in 可迭代对象:
  循环体语句(块)1
else:
  语句(块)2
```

或者：

```
while 条件表达式 :
    循环体语句(块)1
else:
  语句(块)2
```

习题

一、选择题

1. 以下关于循环结构的描述，错误的是(　　)。

A. 遍历循环使用 for <循环变量> in <循环结构>语句，其中，循环结构不能是文件

B. 使用 range()函数可以指定 for 循环的次数

C. for i in range(5)表示循环 5 次，i 的值是从 0 到 4

D. 用循环结构遍历字符串时，循环的次数是字符串的长度

2. 下面代码的输出结果是(　　)。

```
for n in range(400,500):
    i = n // 100
    j = n // 10 % 10
    k = n % 10
    if n == i ** 3 + j ** 3 + k ** 3:
      print(n)
```

A. 407　　B. 408　　C. 153　　D. 159

3. 以下程序的输出结果是(　　)。

```
for i in "Summer":
if i == "m":
  break
  print(i)
```

A. M　　B. mm　　C. mmer　　D. 无输出

4. 以下程序的输出结果是(　　)。

```
for i in "the number changes":
 if i == 'n':
   break
 else:
   print( i, end = "")
```

A. the umber chages　　B. thenumberchanges

C. theumberchages　　D. the

5. 以下关于分支和循环结构的描述，错误的是(　　)。

A. Python 在分支和循环语句里使用类似 x <= y <= z 的表达式是合法的

B. 分支结构中的代码块是用冒号来标记的

C. while 循环如果设计不小心会出现死循环

D. 双分支结构的<表达式 1 > if <条件> else <表达式 2 >形式，适合用来控制程序分支

6. for 或者 while 与 else 搭配使用时，关于执行 else 语句块描述正确的是(　　)。

A. 仅循环非正常结束后执行(以 break 结束)

B. 循环正常结束后执行

C. 总会执行

D. 永不执行

二、填空题

1. 在 Python 无穷循环“while True：”的循环体中，可以使用_____语句退出循环。

2. Python 语句“for i in range(10)：print(i，end＝' ')”的输出结果是_____。

3. 循环语句 for i in range(－3，21，4)的循环次数为_____。

4. 要使语句 for i in range(_____，－4，－2)循环执行 15 次，则循环变量 i 的初值应当为_____。

5. 执行下列 Python 语句后的输出结果是_____，循环执行了_____次。

```
i = -1;
while (i < 0):  i *= i
print(i)
```

6. 在循环结构中，可以使用_____语句结束当前循环。

7. 迭代器是一个对象，表示可迭代的数据集合，包括方法_____和_____，可实现迭代功能。

三、编程题

1. 写出下列 Python 语句的运行结果。

```
x = 2; y = 2.0
if(x == y): print("Equal")
else: print("Not Equal")
```

2. 阅读下面的 Python 程序，该程序的功能是什么？

```
import math
n = 0
for m in range(101, 201, 2):
  k = int(math.sqrt(m))
  for i in range(2, k+2):
    if m % i == 0: break
  if i == k+1:
    if n % 10 == 0: print()
    print('%d' % m, end = ' ')
    n += 1
```

3. 编写程序，计算 1＋2＋3＋…＋100 之和。

4. 编写程序，打印九九乘法表。要求以下三角的方式输出九九乘法表。

5. 输入任意实数 x，根据以下近似公式计算 e^x 的近似值，直到最后一项的绝对值小于 10^{-6} 为止。

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$$

6. 编写程序，输出第一个大于 100 的 3 的幂次方。

7. 编写程序，分别计算 1～100 中所有奇数的和及偶数的和。

8. 计算一个正实数 a 的平方根可以使用牛顿迭代法实现：首先假设 t＝a，开始循环。如果 t＝a/t，则 t 等于 a 的平方根，循环结束并返回结果；否则，将 t 和 a/t 的平均值赋值给 t，继续循环。编写程序，使用牛顿迭代法求解平方根。

9. 用户任意输入 5 个整数，编写程序，输出 5 个数中的最大值。

10. 编写程序，随机生成 100 个[1，100]内的整数，并输出其中的最大值。

11. 编写程序，判断所输入的任意一个正整数是否为质数。

12. 编写程序，输出 200～300 之间的质数，要求每行输出 5 个质数。