第3章

程 序 结 构

一个程序由一系列语句构成，其中的语句可分为简单语句和复合语句。简单语句包括赋值语句、输入输出语句和表达式语句等。复合语句包括条件语句和循环语句等，按照一定的方式控制简单语句的运行。

程序结构分为 3 种：顺序结构、分支结构和循环结构。

3.1 顺序结构

一个程序由一系列语句构成，程序运行时按照语句顺序逐个执行。例如，以下是包括输入、输出和赋值语句的程序：

```
"""This is a one line docstring: Compute BMI for a person"""
""" This is a multi-line docstring:

    input weight in kg and height in m,
    compute BMI and print it out.
"""
w = float(input('type your weight in kg:')) # 提示用户输出体重
h = float(input('type your height in m:'))  # 提示用户输入身高
bmi = w / h**2                              # 计算bmi
print('Your BMI is', round(bmi,1))          # 显示结果
```

以上程序是包含 4 个简单语句的顺序结构程序。一个顺序结构程序中的语句也可以是更复杂的语句，如后面将看到的条件语句或者循环语句。

简单语句通常由一行构成。复合语句通常由多行构成。本节介绍几种常用的简单语句。

3.1.1 简单赋值语句


赋值语句

一个**赋值语句** (assignment statement) 的作用是设置或者修改一个变量的值，其伪代码格式为 var ← expr，在 Python 中的格式为

```
var = expr
```

其中 var 是变量名，expr 是一个表达式。该语句被执行时，先计算 expr 的值，然后令 var 表示该值，或者说 var 指向 expr 表示的对象，或者说将 expr 的值绑定于 var。例如，图 3.1 表示赋值语句 x = 3 完成后，x 指向数据对象 3。

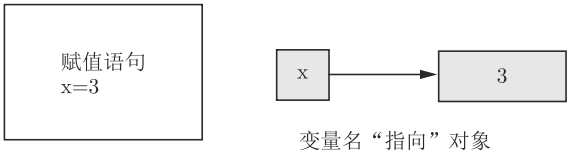


图 3.1　赋值语句及其内存

注意，这里的 var 可以是任何符合变量命名规则的标识符。程序设计语言通常规定任何字母或者下画线开始的字母、数字和下画线构成的串都可以作为变量名，但保留字除外，而且中间不能有空格。

不同变量可以指向同一个对象，如图 3.2 所示。一个变量的值可以被重置，如图 3.3 所示。

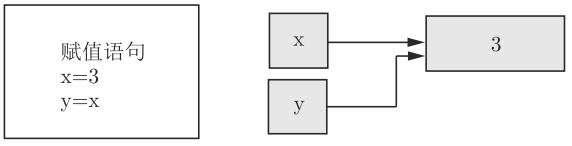


图 3.2　不同变量指向同一个对象

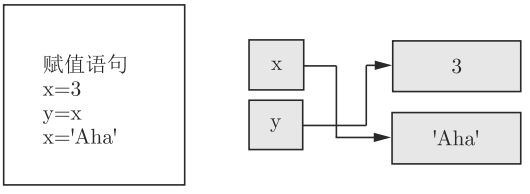


图 3.3　重置一个变量的值

那么下列语句序列执行完后，变量 x 和 y 的值各是什么呢？

```
>>> x = 3
>>> y = x
>>> x = x + 1
```

结果是，x 的值为 4，y 的值仍然是 3。赋值语句 x = x + 1 并没有修改 x 原指向的对象 3，而是构造了一个新对象 4，并让 x 指向新的对象，而 y 的值不变，如图 3.4 所示。

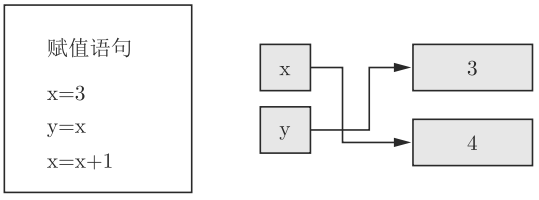


图 3.4　不可变对象的赋值

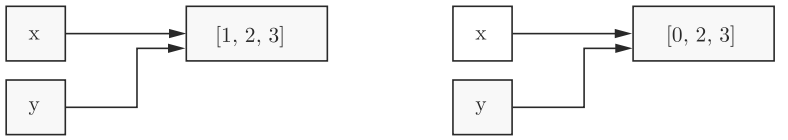
3.1.2 可变对象与别名

一个变量赋值给另一个变量时，两个变量指向同一个对象，称为**别名** (alias)。例如，图 3.2 中的两个变量 x 和 y 同时指向数据 3。

再看以下赋值语句序列：

```
>>> x = [1, 2, 3]
>>> y = x
>>> x[0] = 0
>>> y
[0, 2, 3]
>>> x
[0, 2, 3]
```

第二个赋值语句使得 x 和 y 同时指向可变列表对象 [1,2,3]。在这种情况下，赋值语句 x[0]= 0 修改了该对象的局部，所以，x 和 y 均指向修改后的对象。图 3.5(a) 显示第二个赋值语句`y=x` 执行之后的变量状态，图 3.5(b) 显示第三个赋值语句 `x[0]=0` 执行之后的变量状态。



(a) 两个变量指向同一个列表　　(b) 通过变量x修改了x和y共同指向的列表

图 3.5　通过一个变量可以修改多个变量共同指向的列表

可变对象与别名

需要注意的是，如果这种别名现象并不是我们想要的，那么应该使用列表的副本，使得修改列表的副本不会改变原有的列表。这种别名现象，特别是所指的对象是列表这样的可变对象时，有可能导致难以发现的错误（参见 5.1.3 节）。

3.1.3 复合赋值

形如 $x = x + e$ 的赋值语句是一种常见的形式，它在变量 x 当前值基础上加上表达式 e 的值。对于这一类赋值语句，Python 支持简化的**复合赋值**（compound assignment），或称增强赋值（augmented assignment），写为 `x += e`。例如：

```
>>> x = 0
>>> x += 1
>>> print(x)
1
```

赋值 `x += 1` 称为复合加法赋值，`+=` 称为复合加法运算符。类似地，对于其他的二元运算也可以使用相应的复合赋值，见表 3.1。

表 3.1　复合赋值运算

复合赋值运算符	名称	例	等效的赋值语句
+=	复合加法赋值	i += 1	i = i + 1
-=	复合减法赋值	i -= 1	i = i - 1
*=	复合乘法赋值	i *= 2	i = i*2
/=	复合浮点除法赋值	i /= 2	i = i / 2
//=	复合整数除法赋值	i //= 3	i = i // 3
%=	复合求模赋值	i %= 3	i = i % 3
**=	复合幂赋值	i **= 2	i = i ** 2

3.1.4　并行赋值

Python 支持同时给多个变量赋值，即**并行赋值** (simultaneous assignment)。例如

```
>>> x, y = 1, 2
>>> print(x)
1
>>> print(y)
2
>>> x, y = x + 1, x + y
>>> print(x)
2
>>> print(y)
3
```

并行赋值同时计算赋值号右边表达式的值，并从左到右将左边变量分别绑定对应表达式的值。

并行赋值还可以用于交换两个变量的值。例如：

```
>>> x, y = 1, 2
>>> y, x = x, y
>>> print(x,y)
2 1
```

3.1.5　输入和输出语句

输入、输出和赋值是几个最简单的语句。输入语句 `input()` 用于接收用户在键盘输入的信息，其语法为

```
input([prompt])
```

语法中的方括号 [] 表示其中的实参是可选的，即 input() 可以不带参数，也可以带参数。如果带参数，该参数是一个字符串，用于提示（prompt）用户输入。在执行 `input()` 时，系统等待用户从键盘输入，直至用户按回车键，然后函数将用户输入的字符串返回，即函数 input() 的返回值是用户输入的字符串（不包括回车符）。例如：

```
>>> name = input('What is your name:')
What is your name: 高兴
```

```
>>> name
' 高兴'
>>> age = input('How old are you:')
How old are you:20
>>> age
'20'
```

注意，age 是一个字符串。但是，这里希望 age 的类型为 int，所以，通常在 input() 外嵌套转换函数 int()：

```
>>> age = int(input('How old are you:'))
How old are you:20
>>> age
20
```

输出语句 print() 用于将信息显示到屏幕上。

输出语句 print() 的语法为

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

它表示 print() 可以有多个打印参数 (*object 表示可以提供多个实参)，后面的 4 个实参可以省略。如果省略，则将其中的输出参数逐个输出到标准输出文件（即屏幕，file=sys.stdout），两个单引号中间用空格分离 (sep=' ')，最后换行(end='\n')。例如，用一个 print() 打印多个参数信息：

```
>>> name = ' 高兴'
>>> print('My name is', name, "and I'm", 20)
My name is 高兴 and I'm 20
```

也可以改变分隔符。例如：

```
>>> name = ' 高兴'
>>> print('My name is',name, "and I'm", 20,sep=' : ')
My name is : 高兴 : and I'm : 20
```

可以用多个 print() 语句打印参数信息。例如：

```
>>> name = ' 高兴'
>>> print('My name is'); print(name)
My name is
高兴
```

其中语句后的分号表示一个语句结束，这里用于表示连续执行两个语句。

可以看到，两个 print() 语句分别打印在两行。如果希望将两个 print() 的信息显示在同一行，可以改变命名参数 end 的值。例如：

```
>>> name = ' 高兴'
>>> print('My name is',end=' '); print(name)
My name is 高兴
```

【注意】 Python 中的许多函数和方法都可以带不同个数的参数，即同一个名有不同用法，如这里的 input() 和 print()、第 2 章的函数 range()、字典上的查找方法 get() 和删除方法 pop() 等。在程序设计中，这种用同一个名实现不同用法的方法称为**重载** (overloading)。

3.1.6　表达式语句

表达式语句 (expression statement) 由一个或者多个表达式构成，表达式之间用逗号分隔。例如，小海龟作图的命令：

```
turtle.penup(), turtle.forward(100)
```

表达式语句中，每个表达式通常是一个方法或者函数的调用，其目的是完成某个操作，而不是计算一个值（如 math.sqrt() 只是计算一个值）。主要任务是完成某种操作而不是计算一个值的函数也称为**过程** (procedure)。表达式语句通常用于过程调用。

3.2　分支结构

分支结构用于控制在不同的条件下执行不同的语句。分支结构包括单分支 if、双分支 if-else 和多分支 if-elif-else 等条件语句。


分支结构

3.2.1　if 语句

如果只有在某种条件 be 满足时才执行 S1，则使用 if 条件语句。图 3.6 给出带上下文的 if 语句的一般形式和**程序框图**（flowchart，也称程序运行流程图，简称流程图），即条件语句前后分别添加了语句 S0 和 S2。条件语句中 be 是一个布尔表达式，S0、S1 和 S2 是一个语句或者多个语句构成的语句块。只有在表达式 be 的值计算为 True 时，S1 才会执行。S1 由多个语句构成时，要缩进左对齐。

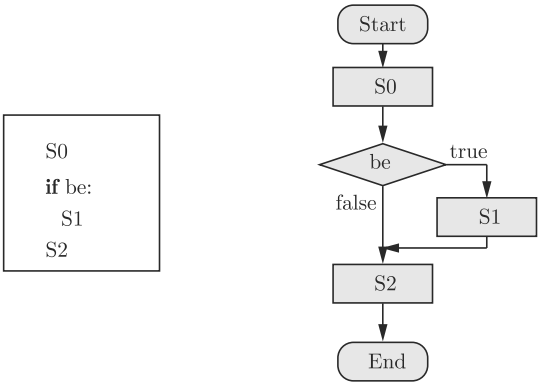


图 3.6　带上下文的 if 语句的一般形式和程序框图

例如，设计一个程序，输入一个人的体重和身高，然后输出其 BMI。如果 BMI 小于 25，则输出“体重正常”字样，见程序 3.1。

程序 3.1　计算 BMI

```
weight = float(input('Input your weight (kg):'))
height = float(input('Input your height(m):'))
bmi = weight / height**2
print('Your BMI is',round(bmi,2))
if bmi < 25:
    print('体重正常')
```

尝试输入不同的体重和身高，查看其运行结果。

【注意】 Python 解释器默认脚本使用编码 UTF-8 存储。在存储脚本时应选择编码 UTF-8，以便解释器正确识别汉字。否则，对于包含汉字的脚本，Python 解释器可能报错，如“SyntaxError: Non-UTF-8 code starting with ... , but no encoding declared”。一种解决方法是重新用 UTF-8 编码存储文件，另一种解决方法是在脚本第一行用注释说明脚本编码，如 `#coding=gbk`。

3.2.2　if-else 语句

在计算并输出 BMI 的程序中，如果 bmi 变量的值小于 25，则输出“体重正常”，否则输出“有点超重哦:-)”，见程序 3.2。

程序 3.2　计算 BMI

```
weight = float(input('Input your weight(kg):'))
height = float(input('Input your height(m):'))
bmi = weight / height**2
print('Your BMI is',round(bmi,2))
if bmi < 25:
    print('体重正常')
else:
    print('有点超重哦:-)')
```

图 3.7 给出带上下文的 if-else 语句一般形式和程序框图。

3.2.3　if-elif-else 语句

带上下文的多分支条件语句一般形式和程序框图如图 3.8 所示。

例如，在计算并输出 BMI 的程序中，如果 bmi 变量的值小于 25，则输出“体重正常”；如果 bmi 变量的值介于 25 和 30 之间，则输出“有点超重哦:-)”；如果 bmi 变量的值大于 30，则输出“严重超重哦:-)”：

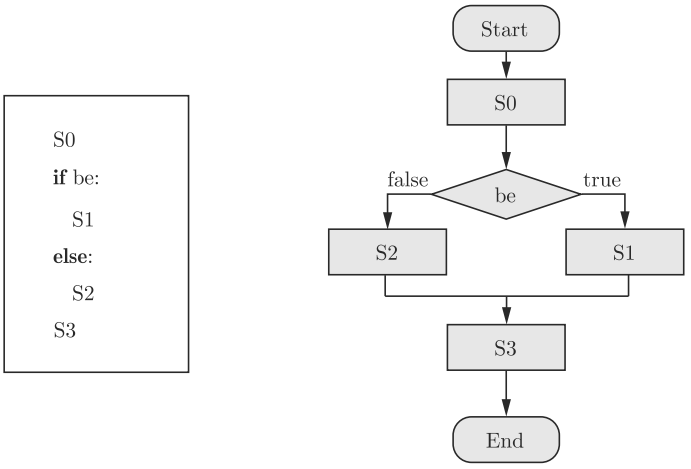


图 3.7　带上下文的 if-else 语句的一般形式和程序框图

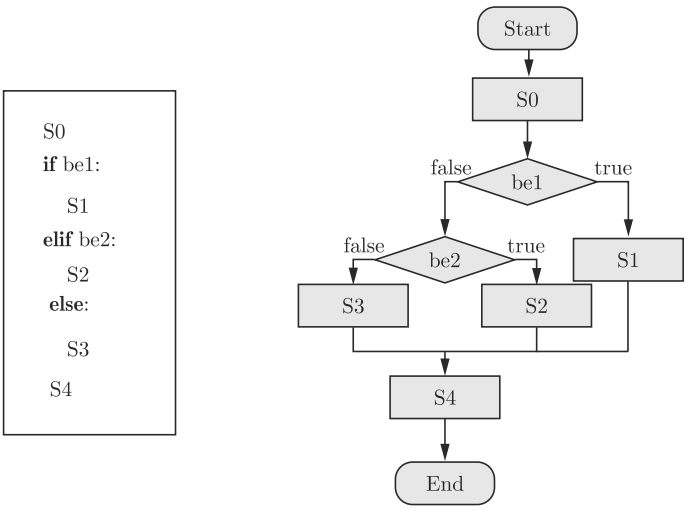


图 3.8　带上下文的 if-elif-else 语句的一般形式和程序框图

```
weight = float(input('Input your weight(kg):'))
height = float(input('Input your height(m):'))
bmi = weight / height**2
print('Your BMI is',round(bmi,2))
if bmi < 25:
   print('体重正常')
elif bmi < 30:
   print('有点超重哦:-)')
else:
   print('严重超重哦:-)')
```

例 3.1　编写模拟买彩票的程序。

(1) 随机生成一个两位数，可用 random 模块的 randint(10,99) 生成。

（2）提示用户输入一个两位数的猜测值。

（3）如果用户的猜测完全正确，则打印“您赢了 10 万元”；如果用户十位数猜对了，或者个位数猜对了，则打印“您赢了 100 元”；如果两位数都猜错了，则打印“您没有对上任何数”。

（4）打印随机数。

这里可以用 random.randint(10,99) 随机生成 10~99 的整数。另外，要取得一个两位整数的个位数和十位数，可以分别用 10 取模和做整数除法得到，见程序 3.3。

程序 3.3　模拟买彩票

```
import random
answer = random.randint(10,99)
guess = int(input('Input some int (10—99):'))
answer0 = answer % 10          # 随机数的个位
answer1 = answer // 10         # 随机数的十位
if guess == answer:
    print('您赢了 10 万元')
elif (guess % 10 == answer0) or (guess // 10== answer1):
    print('您赢了 100 元')
else:
    print('您没有对上任何数')
print('秘密数是',answer)
```


彩票程序
(例3.1)

3.3　循环结构


循环结构

循环语句用于表达某些语句的反复执行。一般程序设计语言均提供两种表示循环的语句：for 循环和 while 循环。当重复的次数确定时，通常使用 for 循环语句；当重复的条件确定时，通常使用 while 循环语句。

3.3.1　for 循环语句

for 循环语句用于表达重复执行某个语句（块）一定的次数。for 循环伪代码与 Python 代码对照见表 3.2。

表 3.2　for 循环伪代码和 Python 代码对照

伪代码	Python 代码	说明
for $i \leftarrow 0$ to $n-1$ 　S	for i in range(n): 　S	注意 Python 代码中的冒号； 循环体 S 要缩进并左对齐

例如，打印下面的图形：

```
#####
#####
```

```
#####
#####
```

完成这个任务的算法是

```
重复执行下列语句4次：
  打印 "#####"
```

或者（n=4）

```
for i = 0 to n-1:
   打印 "#####"
```

所以，Python 程序可以写成

```
for i in range(4):  # i分别取0、1、2、3这4个值
    print("#####")
```

再如，打印 1~10 各数的平方：

```
for i in range(1,11):
    print(i, '*', i, '=', i ** 2)
```

输出结果如下：

```
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
10 * 10 = 100
```

for 语句的一般形式和程序框图如图 3.9 所示。

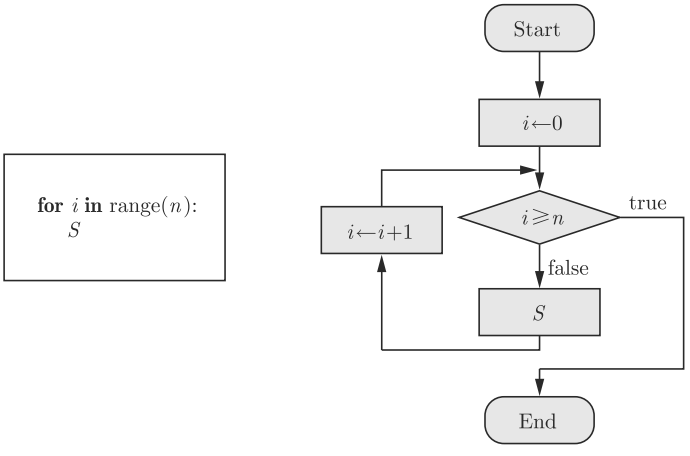


图 3.9　for 语句的一般形式和程序框图

例 3.2 设计一段求第 n 个斐波那契数的程序。输入是 n，输出斐波那契数列 $f_0=0, f_1=1, f_2=1, f_3=2, f_4=3, \cdots$ 中的 f_n。

【方法】 将以上迭代计算方法写成算法。

计算的方法是：每次将现有的两个数相加，得到下一个数：如果用两个变量 f_1 和 f_2 分别表示当前计算出的两个斐波那契数，那么下一个数是 $f_3=f_1+f_2$。为了重复这个关键步骤，需要每次都用 f_1 和 f_2 分别表示当前计算出的两个斐波那契数，因此，在完成计算 $f_3=f_1+f_2$ 后，需要用 f_1 存储 f_2 的值，f_2 存储 f_3 的值，并重复这个过程，直至求得所需的值。

【算法】 由此得到算法 3.1。

算法 3.1 fib(n)

输入: $n \geqslant 2$ 是一个整数

输出: 输出第 n 个斐波那契数

$f_1 \leftarrow 0$
$f_2 \leftarrow 1$
for $i \leftarrow 1$ **to** $n-1$ **do**
  $f_3 \leftarrow f_1+f_2$
  $f_1 \leftarrow f_2$
  $f_2 \leftarrow f_3$
输出 f_3


斐波那契数(例3.2)

【代码】 对于这种形式的伪代码，在学习 Python 函数之后，很容易将其转换为 Python 函数。在使用自定义函数之前，我们只用 Python 函数表达算法，并在算法的第一个语句之前设置输入参数。例如，通过赋值语句或者输入语句 input() 设置输入参数。由此得到程序 3.4。

程序 3.4 求第 n 个斐波那契数

```
n = int(input("输入整数: "))
f1 = 0
f2 = 1
for i in range(1,n):
    f3 = f1+f2
    f1 = f2
    f2 = f3
print("第"+ str(n) +"个斐波那契数 : ", f3)
```

【注意】 程序设计中经常用**迭代** (iteration) 这个术语。类似于数学上的迭代公式，它表示重复某种操作有限次，直至得到所需的数据。迭代也泛指循环。使用迭代设计算法时，最关键的是说明重复的操作以及重复的次数或者重复的条件。

3.3.2 while 循环语句

如果需要在一定的条件下重复执行一个或者一组语句，可以使用 while 循环语句。

while 循环伪代码与 Python 代码对照见表 3.3。

表 3.3　while 循环伪代码和 Python 代码对照

伪代码	Python 代码	说明
while condition 　S	while condition: 　S	注意 Python 代码中的冒号； 循环体 S 要缩进并左对齐

while 循环语句的一般形式和程序框图如图 3.10 所示。

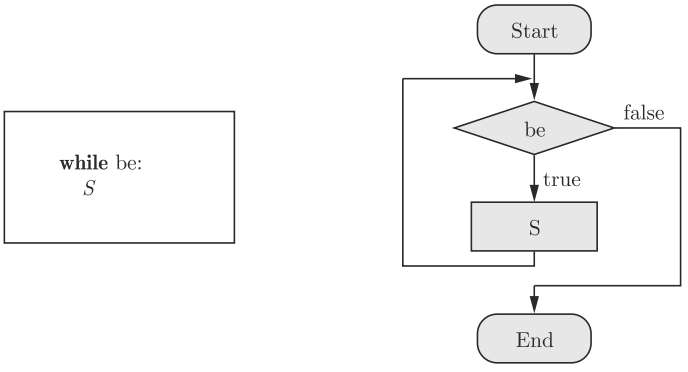


图 3.10　while 循环语句的一般形式和程序框图

例如，输出 1 至 10 的平方根：

```
import math
i = 1
while i <= 10:
    print('squre root of ', i, 'is', round(math.sqrt(i), 2))
    i = i + 1
```

输出结果如下：

```
squre root of  1 is 1.0
squre root of  2 is 1.41
squre root of  3 is 1.73
squre root of  4 is 2.0
squre root of  5 is 2.24
squre root of  6 is 2.45
squre root of  7 is 2.65
squre root of  8 is 2.83
squre root of  9 is 3.0
squre root of  10 is 3.16
```

需要特别注意的是，while 循环中应该保证循环变量的值不断变化，在有限步后使得循环条件不成立，因此终止循环语句。例如，上例中的循环变量 i，如果循环体中没有语句 `i = i + 1`，则每次循环后 i 的值 1 保持不变，因此出现无限循环。

例 3.3 猜数游戏。

问题陈述如下：

（1）程序秘密设定一个 1～50 的数 secret。

（2）程序提示用户输入一个猜测值 answer。

（3）如果 answer 等于 secret，则程序结束，输出“猜中了”字样；否则提示用户猜测的值是大于 secret 还是小于 secret，让用户继续猜，直至猜中。

【方法】 将以上陈述进一步细化：

（1）程序秘密设定一个 1～50 的数 secret，可用 randint(1,50) 随机生成。

（2）程序提示用户输入一个猜测值，并将该输入数值记作 answer。

（3）当 answer 不等于 secret 时，重复下列步骤：

① 提示用户猜测的值是大于 secret 还是小于 secret。

② 让用户继续猜，并仍用 answer 记录用户输入的猜测值。

（4）以上循环结束时，answer 等于 secret，输出“猜中了”字样，程序结束。

【算法】 至此，可以将以上方法写成伪代码形式的算法 3.2。

算法 3.2 guessNumber()

```
secret ← randint(1, 50)          //生成一个 1～50 的秘密数
answer ← 读取用户的猜测值
while answer ≠ secret do
    if answer > secret then
        输出提示信息"猜的大了"
    else
        输出提示信息"猜的小了"
    answer ← 用户的猜测值
输出信息 "猜中了"
```

猜数游戏
(例3.3)

注意，这里重复猜测的次数是不确定的，但是重复的条件是确定的，即当且仅当猜测值等于秘密值的时候才结束猜测，因此使用 while 循环。

【代码】 算法 3.2 的实现见程序 3.5。

程序 3.5 猜数游戏

```
import random
secret = random.randint(1,50)
answer = int(input("输入一个 1～50 的整数："))
while answer != secret:
    if answer > secret:
        print("您这次的猜测大了")
    else:
        print("您这次的猜测小了")
    answer = int(input("请再输入一个 1～50 的整数："))
print("您猜中了！是",secret)
```

例 3.4　用猜数的方法求一个数的平方根，如 $\sqrt{2}$。

【方法】　首先，已知 $1<\sqrt{2}<3$，或者说 $\sqrt{2}\in(a,b)$，其中 $a=1,b=3$。第一次猜区间 (a,b) 的中点 $c=(a+b)/2$。如果 $c^2>2$，则说明 2 的平方根介于 a 和 c 之间，因此到区间 (a,c) 上接着猜；如果 $c^2<2$，则到区间 (c,b) 上接着猜。重复这个过程，直至包含 $\sqrt{2}$ 的区间足够小，或者足够精确时，则可取最后一个包含 $\sqrt{2}$ 的区间中点作为近似值。

将以上求 $\sqrt{2}$ 近似值的二分猜测法进一步细化如下：

（1）设定初始包含 $\sqrt{2}$ 的区间 (a,b) 为 $(1,3)$。

（2）设定一个精度：$e=0.001$。

（3）重复下列步骤，直至 $|b-a|<e$:

① 取区间 (a,b) 的中点 $c=(a+b)/2$。

② 如果 $c^2>2$，则到区间 (a,c) 上接着猜，即 $(a,b)=(a,c)$。

③ 如果 $c^2<2$，则到区间 (c,b) 上接着猜，即 $(a,b)=(c,b)$。

（4）取区间 (a,b) 的中点 $c=(a+b)/2$ 作为 $\sqrt{2}$ 的近似值。

表达循环的关键在于每次循环都用 (a,b) 表达 2 的平方根所在的区间。

【算法】　现在可以将以上方法写成伪代码形式的算法（见算法 3.3）。同样，因为循环的次数不确定，但是循环条件确定，因此使用 while 循环。

算法 3.3　my_sqrt()

输出: 计算 2 的近似平方根

```
(a,b) ← (1,3)
e ← 0.001
while |b − a| ⩾ e do
    c ← (a + b)/2
    if c^2 > 2 then
        (a,b) ← (a,c)
    else
        (a,b) ← (c,b)
输出近似值 (a + b)/2
```

【代码】　容易将算法 3.3 转换为程序 3.6。

程序 3.6　计算 2 的平方根

```
"""Find square root of 2 with precision e"""
e = 0.001                    # 精度
a, b = 1.0, 3.0              # 初始区间
while abs(b-a) >= e:
    c = (a + b) / 2
    if c**2 > 2:
        a, b = a, c
    else:
```

程序 3.6 (续表)

```
        a, b = c, b
    c = (a + b) / 2
    print("2 的平方根近似值：", c)
    # 运行结果为"2 的平方根近似值：1.41455078125"
```

这里用到了并行赋值，即多个赋值语句同时进行：计算赋值语句右边的表达式，然后将其同时赋值给左边的变量。

求平方根收敛最快的方法是牛顿-拉弗森（Newton-Raphson）公式，见习题 3.9。

3.3.3 循环的控制：break 和 continue

在循环体中，可能需要依据某个条件调整循环执行的逻辑。例如，中断循环，或者结束本次循环，进入下一次循环。break 语句将中断包含该语句的循环，转去执行循环语句后的语句。例如：

```
for i in range(1, 10):
    if i ** 2 <= 5:
        print(i ** 2)
    else:
        break
print('break goes to here')
print('i = ', i)
```

在上面代码中，for 循环语句后的第一个语句是与 for 对齐的第一个 print()，因此程序输出结果如下：

```
1
4
break goes to here
i =  3
```

语句 continue 将结束本次循环，并进入下一次循环，即跳过本次循环的剩余语句，进入下一次循环。例如：

```
for i in range(1, 10):
    if i % 2 == 0:
        print(i)
    else:
        continue
    print('current i =', i)
```

在以上代码中，当 i 取奇数时 if 条件不成立，因此跳过条件语句后的 print 语句，进入下一次循环，输出结果如下：

```
2
current i = 2
4
current i = 4
6
current i = 6
8
current i = 8
```

有的算法可能完成循环后终止，也可能在循环过程中终止，即算法有两个出口，例如算法 1.2。对于这种情况，可以使用 Python 的带 else 的 while 循环：

```
while condition:
  S1
else:
  S2
```

其中循环条件 condition 成立时执行语句 S1，不成立时执行 S2。如果语句块 S1 包含 break 语句，则执行 break 后终止循环，不再执行语句 S2。

循环可能有两种终止方式（在循环体中终止和在循环条件不成立后终止），中间可能终止的 while 循环伪代码和 Python 代码对照见表 3.4。

表 3.4　中间可能终止的 while 循环伪代码和 Python 代码对照表

伪代码	Python 代码	说明

while condition1
　S1
　if conditon2
　　S2
　　STOP
　S3
S4
	while condition1:
	　S1
	　if condition2:
	　　S2
	　　break
	　S3
	else:
	　S4
		注意 Python 代码中的冒号；
		语句块要缩进并左对齐

例 3.5　判断任意一个正整数是否素数。

【方法】　判断一个正整数 n 是否素数的基本方法：依次检查 $2, 3, \cdots, n-1$ 是否为 n 的因子，如果其中一个数是因子，则输出“合数”并终止，否则输出“素数”并终止。

【算法】　循环选择 for 和 while 均可。本例的算法如算法 3.4 所示。

【代码】　因为算法可能在检查到因子时终止，而且不执行 while 循环后的语句，因此使用了带 else 的循环。将算法转换为带 else 的循环的 Python 代码见程序 3.7。


判断素数
(例3.5)

程序 3.7　判断一个正整数是否素数的代码

```
n = int(input('input some int:'))
factor = 2
```

程序 3.7 (续表)

```
    while (factor < n):
        if (n % factor == 0):
            print(n, 'is not a prime')
            break
        else:
            factor += 1
    else:
        print(n, 'is a prime')
```

算法 3.4 isPrime(n)

输入: n 是正整数，$n \geqslant 2$

输出: 如果 n 是素数，则输出“素数”，否则输出“合数”

```
factor ← 2
while factor ⩽ n − 1 do
  if n%factor = 0 then
    输出"合数"
    算法终止
  else
    factor ← factor + 1
输出 "素数"
```

对于以上判断素数的例子，另一种方法是设置一个布尔型标记变量 `is_prime`，根据该变量的值决定是否结束循环以及循环结束后执行的语句。例如，程序 3.8 中，将变量 `is_prime` 初始化为 `True`。在检测 `n` 是否有因子的循环中，一旦发现有因子，则将 `is_prime` 置为 `False`，并终止循环（break）。在循环之后，根据 `is_prime` 的值打印相应信息。

程序 3.8 使用一个布尔标记判断一个正整数是否素数的代码

```
n = int(input('input some int:'))
factor = 2
is_prime = True
while (factor < n):
    if (n % factor == 0):
        is_prime = False
        break
    else:
        factor += 1
if is_prime:
    print(n, 'is a prime')
else:
    print(n, 'is not a prime')
```

3.3.4　循环嵌套

循环可以嵌套，即循环体中的语句也可以是一个 for 循环或者 while 循环。

例如，打印如图 3.11 所示的九九乘法表，可以使用如下代码：

```
for i in range(1,10):
    for j in range(1,10):
        print(format(i * j, '4d'), end = ' ')
    print()
```

其中外循环的循环体包含两个语句，第一个语句也是一个 for 循环，第二个语句是一个简单的 print 语句。外循环控制行，内循环打印各行信息。

```
   1    2    3    4    5    6    7    8    9
   2    4    6    8   10   12   14   16   18
   3    6    9   12   15   18   21   24   27
   4    8   12   16   20   24   28   32   36
   5   10   15   20   25   30   35   40   45
   6   12   18   24   30   36   42   48   54
   7   14   21   28   35   42   49   56   63
   8   16   24   32   40   48   56   64   72
   9   18   27   36   45   54   63   72   81
```

图 3.11　九九乘法表


循环嵌套

为了将每 9 个乘积显示在同一行，并各列对齐，需要说明各个乘积占用的宽度。这里使用了**格式**（formatting）**函数** `format(value, format_spec)`。其中，第一个参数 value 表示打印的值；第二个参数 format_spec 是字符串，表示打印格式，包括对齐方式、占用宽度、精度和类型等。这里，`format(i * j, '>4d')` 表示乘积 `i * j` 将占 4 位，右对齐（用 `>` 表示），d 表示第一参数的类型是整数。下面给出打印数值和字符串的典型例子：

- 打印整数 123456（用 d 表示整数类型），左对齐（用 `<` 表示），占用 10 位，空白位置用#填充：

```
>>> print(format(123456,'#<10d'))
123456####
```

- 打印浮点数 123.456（用 f 表示浮点数类型），右对齐（用 `>` 表示），占用 10 位，其中小数点后保留 2 位 (用.2 表示)：

```
>>> print(format(123.456,'>10.2f'))
    123.46
```

打印数值右对齐时，可以省去对齐符号 `>`。

- 打印字符串`'Python Programming'`（用 s 表示字符串类型），居中对齐（用^表示），占用 30 位：

```
>>> print(format('Python Programming', '^30s'))
      Python Programming
```

此外，打印语句中的 end 参数表示该打印语句末尾不换行，而是用一个空格结尾。这样便可以用内循环将九个乘积输出在同一行，外循环的第二个语句 print() 达到换行目的。

如果在九九乘法表外围打印一些装饰，则可以产生更完整、美观的九九乘法表。例如：

```
print(format("九九乘法表\n", '^40s'))  # 标题字符串居中
print("i*j | ", end = '')
for j in range(1, 10):
    print("  ", j, end = ' ')         # 打印表头
print()                               # 换行
print("-"*50)

# 显示乘法表
for i in range(1, 10):
    print(format(i, '2d'), " |", end = ' ')
    # 内循环将9个乘积显示在同一行
    for j in range(1, 10):
        print(format(i * j, '4d'), end = ' ')
    print()                           # 换行
```

以上程序打印数值时采用右对齐格式，因此省去了右对齐符号 >。程序输出见图 3.12。

九九乘法表

```
i*j |    1    2    3    4    5    6    7    8    9
----------------------------------------------------
 1  |    1    2    3    4    5    6    7    8    9
 2  |    2    4    6    8   10   12   14   16   18
 3  |    3    6    9   12   15   18   21   24   27
 4  |    4    8   12   16   20   24   28   32   36
 5  |    5   10   15   20   25   30   35   40   45
 6  |    6   12   18   24   30   36   42   48   54
 7  |    7   14   21   28   35   42   49   56   63
 8  |    8   16   24   32   40   48   56   64   72
 9  |    9   18   27   36   45   54   63   72   81
```

图 3.12　加上行列号的九九乘法表

习题

3.1　设计一个计算一元二次方程 $ax^2 + bx + c = 0$ 的根的程序。输入分别是 3 个实数 a、b 和 c，根据判定式打印两个实根，或者打印“不存在实根”。

3.2　设计程序，用户输入一个课程成绩（score），程序根据表 3.5 判断并输出等级。

表 3.5　成绩范围与等级

成绩范围	等级
score ⩾90	A
80⩽ score < 90	B
70 ⩽ score <80	C
60 ⩽ score < 70	D
score < 60	F

3.3　参照《中华人民共和国个人所得税法》最新个人所得税率，设计一个计算个人所得税的程序。

3.4　编写程序打印每边由 n 个 * 构成的菱形图案。要求第一个语句给变量 n 赋一个初值，例如：

```
n = 10
```

接下来的语句将直接使用 n，而不是使用 10，这样只需要修改此赋值语句右边的常数即可打印每边由任意 n 个 * 构成的菱形图案。

3.5　分别用 for 和 while 循环编写一段程序，判断一个正整数是否素数。要求：提示用户输入一个正整数，然后打印该数是否素数的信息。

3.6　编写程序，对于任意给定的正整数 n，在 turtle 中画出正 n 边形。类似于习题 3.4，要求第一个语句给 n 赋一个具体的值，如 $n = 5$，接下来的语句将使用 n 而不是具体的值（如 5）表达角度，这样只需修改第一个赋值语句右边的数值，即可画出指定的正 n 边形。

提示：正 n 边形内角和为 $180(n-2)$，由此可计算每个内角及外角度数。

3.7　设计一个模拟猜拳（石头剪刀布）的游戏模拟程序。设想有两个玩家，每个玩家（随机）出一个手势，由程序判断这一轮谁赢。然后让玩家继续出手势，直至分出胜负，例如三局两胜。

例如，程序的输出可能是这样的：

玩家 A 出“剪刀”
玩家 B 出“布”
玩家 A 胜，1-0
玩家 A 出“剪刀”
玩家 B 出“布”
玩家 A 胜，2-0
玩家 A 出“剪刀”
玩家 B 出“剪刀”
平手，2-0
玩家 A 出“剪刀”