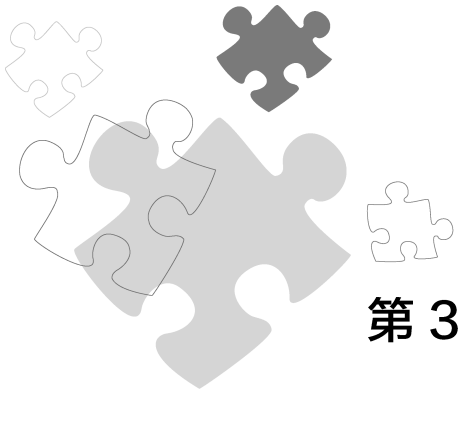

第3章　顺序结构

Python的一个程序是由若干条语句组成的，这些语句用来完成一定的操作任务。语句中包含各种成分，如常量、变量、表达式、函数等。

程序中语句的基本结构包括顺序结构、选择结构和循环结构。按照语句在程序中出现的先后次序依次执行的结构为顺序结构。根据条件选择执行不同语句的结构为选择结构。根据条件重复执行相关语句的结构为循环结构。

3.1　Python程序的编码规范

3.1.1　一个Python程序

下面是一个Python程序，用于展示Python的编码的基本规范。程序中涉及的语法元素会在后面各章中陆续学习，本节不一一介绍。

【例3-1】 一个Python程序。

```
#1.  '''多行注释的第一行
#2.      多行注释的第二行
#3.      多行注释的第三行'''
#4.  def exam(h):
#5.      x = eval(input("输入第一个数值: "))      #输入字符串,并用eval()函数转换
#6.      y = eval(input("输入第二个数值: "))
#7.      s = 0;    a = 1;    b = 2                 #变量s、a、b赋值为0、1、2
#8.      # if语句根据x值选择执行不同的语句块
#9.      if x > 0:                                 #当x大于0时,执行#10～#14行代码
#10.         while h >= 0:                         #while循环语句反复执行#11、#12行代码
#11.             s = h + x
#12.             h = h - 1
#13.         y += h
#14.         print(y)
#15.     else:                                     #当x不大于0时,执行#16行代码
#16.         print(x + y - h)
```

```
#17.        return h                #函数结束,并返回变量 h 的值
#18.
#19.  print('本例演示缩进,本行为程序运行的首行语句')
#20.  a = exam(5)                   #调用自定义函数 exam()
#21.  print(a)                      #输出变量 a 的值
```

运行程序,并且第一个数据输入 4.6,第二个数据输入 3.2,得到输出结果如下:

```
本例演示缩进,本行为程序运行的首行语句
输入第一个数值:4.6↙
输入第二个数值:3.2↙
2.2
-1
```

上面输出结果中的第二行是 input()函数执行时的提示,带下画线部分的内容 4.6↙是运行时敲击键盘输入的内容,行末的符号↙表示的是输入结束时所按的 Enter 键。本书后面的写法与此相同。

例 3-1 的代码本身没有太多的实际用意,只是为了说明 Python 程序的编码基本规范。Python 的程序书写有严格的语法和文法要求,用户录入时如果有一点点的违规,例如大小写写错、少了一个字符或符号、符号录入成全角字符等,Python 的解释器都会报错。

3.1.2 注释

注释是对代码的说明信息,注释的存在与否不影响程序的功能,但建议在代码中增加适量的注释,以提高程序的可读性。注释有以下两种方式。

- 行末注释:在语句的行末增加#,并在其后添加注释的内容。
- 块注释:以三个单引号(''')或三个双引号(""")开头和结尾。

例 3-1 中的#1～#3 行是一个块注释;#5、#7、#9、#10、#15、#17、#20、#21 行是行末注释,在已有语句的末端添加注释,#8 行也是行末注释,但是并未包含有效语句,这也是#注释合法的用法。

在 IDLE 窗口中按 Alt+3 快捷键可以将当前行和所选代码块变为行末注释,按 Alt+4 快捷键可以取消行末注释。

3.1.3 缩进

在 Python 中,用不同的缩进表明代码块之间不同的层次关系,如图 3-1 所示。缩进默认采用 4 个空格,也可以由用户自己指定,但需保持同层代码块前的空格数量相同。缩进也可以使用 Tab 键,但不能是空格和 Tab 键混用。

3.1.4 语句续行与分隔

Python 程序中的语句通常是一行写一条语句。也可以将多条语句写在一行上,此时需要在多条语句之间加上分号(;),但行末无须添加分号。例 3-1 中的#7 行就是一行中包含了三条赋值语句。

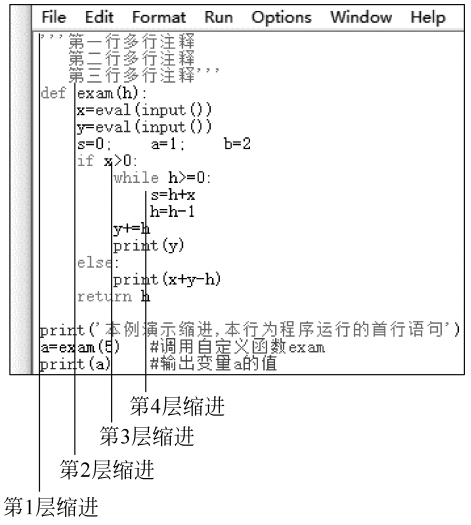

图 3-1　缩进与代码块间的关系

当 Python 的一条语句太长时，也可以将一条语句分成多行来写，只需在该语句的每行后使用反斜杠(\)，但最后一行末无须添加反斜杠。

若在语句中的圆括号(())、方括号([])、花括号({})的内部分行，则不需要使用换行符，三引号之间的内容也可以不加换行符，直接分多行书写。例如：

```
if a == 1 and b == 2 and \
   c == 3 and d == 4 and e == 5:      # 此处可添加注释,上一行的\后不能添加行末注释
    print('OK')
# 以下分多行写,是为了让字典的内容看起来更清楚明了
# 因是在花括号的内部分行,可以不使用续行符
PCounts = {'bcm':0,
          'bcf':0,
          'dcm':0,
          'dcf':0,
          'mnm':0,
          'mnf':0,
          'family':0,
          'famcount':0,
          'Normal':0,
          'Error':0}
```

3.2　变 量 赋 值

3.2.1　赋值语句

1. 赋值的基本格式

赋值是建立变量和数据、对象、函数等之间的联系，赋值后就能通过变量使用数据、对象和函数。赋值的基本格式：

```
var = obj
```

var 是变量，等号的右侧可以是常量、变量、表达式、对象和函数等。例如：

```
>>> a = 1
>>> b = 2
>>> c = 3
```

上述三行语句，对变量 a、b、c 分别赋值为 1、2、3。

2. 复合赋值

复合赋值是指将其他运算与赋值结合在一起。复合赋值包括＋＝、－＝、＊＝、\＝、\\＝、%＝、＊＊＝、<<＝、>>＝、&＝、|＝、^＝。例如：

```
>>> x = 5
>>> x += 1              #等价于 x = x + 1
>>> y = -1
>>> y *= x + 0.5        #等价于 y = y * (x + 0.5)
>>> x, y
(6, -6.5)
```

注意：进行复合赋值运算时，先计算等号右侧的表达式的值，再与左侧的变量进行运算，最后结果存回左侧的变量中。

3. 多变量赋值

1）链式赋值

格式：

```
var1 = var2 = var3 = … = 表达式
```

链式赋值用于将同一个值赋给多个变量。例如：

```
>>> a = b = c = 100
>>> a, b, c
(100, 100, 100)
```

2）同步赋值

格式：

```
var1[, var2[, var3…]] = 表达式 1[, 表达式 2[, 表达式 3…]]
```

注意，等号左右两侧的变量数和表达式数要一致，然后，按位置将表达式值分别赋值给变量。例如：

```
>>> a, b, c = -2.3, 12, 'ab'
>>> a
-2.3
>>> b
```

```
12
>>> c
'ab'
```

同步赋值时，先计算出所有表达式的值后，再赋值给左侧的变量。

另外，Python 的同步赋值可以不借助第三个变量实现两个值的交换操作。例如，下面的两种方法都可以实现 a、b 两个数的内容交换。

方法一：

```
>>> a = 10
>>> b = 9.123
>>> t = a
>>> a = b
>>> b = t
>>> a
9.123
>>> b
10
```

方法二：

```
>>> a = 10
>>> b = 9.123
>>> a, b = b, a
>>> a
9.123
>>> b
10
```

方法一是借助了第三个变量 t 的实现方法，方法二则是采用同步赋值的方法，直接实现了两个数的交换。

3.2.2　变量的共享引用*

用赋值操作“=”将一个变量赋值给另一个变量时，要注意两个变量的值的变化和它们所引用空间的共享现象。本节讨论共享引用中的情况，不是为了把问题复杂化，而是在 Python 中改变变量值时的情况本身就是复杂的。编程者应该了解当前程序中采用的语句会引起变量值发生什么改变。切记：不能想当然地认为是某种结果，建议在使用某些语句前进行一定测试后，再写入到程序中。只有编程者清楚改变值会带来什么结果，程序才会有正确的运行结果。

1. 数值类型和字符串的共享引用

当变量所赋的值为不可变数据类型，将该变量的值赋给别的变量时，多个变量都会指向同一个数据。下面是各种数值类型和字符串等不可变数据类型的共享引用示例。

整数 int 的共享引用示例：

```
>>> x = 1
>>> y = x
>>> x, y
(1, 1)
>>> id(x) == id(y)
True
>>> x is y
True
>>> x = 90
>>> x, y
(90, 1)
>>> id(x) == id(y)
False
```

实数 float 的共享引用示例：

```
>>> x = 3.1238
>>> y = x
>>> x, y
(3.1238, 3.1238)
>>> id(x) == id(y)
True
>>> x is y
True
>>> x = 5.678
>>> x, y
(5.678, 3.1238)
>>> id(x) == id(y)
False
```

复数 complex 的共享引用示例：

```
>>> x = 3 + 4j
>>> y = x
>>> x, y
((3 + 4j), (3 + 4j))
>>> id(x) == id(y)
True
>>> x is y
True
>>> x =  - 2 - 9j
>>> x, y
(( - 2 - 9j), (3 + 4j))
>>> id(x) == id(y)
False
```

字符串 string 的共享引用示例：

```
>>> x = 'Hi!'
>>> y = x
>>> x, y
('Hi!', 'Hi!')
>>> id(x) == id(y)
True
>>> x is y
True
>>> x = x + 'Bye!'
>>> x, y
('Hi!Bye!', 'Hi!')
>>> id(x) == id(y)
False
```

2. 元组 tuple 的共享引用

元组被称为只读列表，即数据可以被查询，但不能被修改。当各个元组值是不可变数据类型时，其共享引用的方式与数值数据类型情况相同。元组中的元素是不可变数据的示例：

```
>>> x = (1, 2, 3)
>>> y = x
>>> x, y
((1, 2, 3), (1, 2, 3))
>>> x == y
True
>>> id(x) == id(y)
True
>>> x is y
True
>>> x = (6,7,8)
>>> x, y
((6, 7, 8), (1, 2, 3))
>>> x == y
False
>>> id(x) == id(y)
False
>>> x[1] = 100
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    x[1] = 100
TypeError: 'tuple' object does not support item assignment
```

若在元组中的元素是可变数据时，则该元组元素的数据值是可变的，但变量指向的元组空间没有变化。这是因为 Python 在元组的存储空间中存放的是数据的引用而不是数据值本身。元组中的元素有可变数据的示例：

```
>>> x = (1, 3, [10, 11])
>>> y = x
>>> x, y
```

```
((1, 3, [10, 11]), (1, 3, [10, 11]))
>>> x == y
True
>>> id(x)
1501033441536
>>> id(x) == id(y)
True
>>> x is y
True
>>> a[0] = 100
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    a[0] = 100
IndexError: list assignment index out of range
>>> x[2][0] = 99
>>> x, y
((1, 3, [99, 11]), (1, 3, [99, 11]))
>>> x == y
True
>>> id(x) == id(y)
True
>>> id(x)
1501033441536
>>> x is y
True
```

图 3-2 中左边的示意图展示了上面元组是不可变数据的示例代码中第一次赋值后 x 和 y 的变量引用情况，右边的示意图展示了上面元组是可变数据的示例代码中的变量引用情况。其中，灰色的列表值“10 的地址”已经变为了“99 的地址”，从而引发了显示元组时值的改变，但元组空间本身并没有改变，因此元组值的变化是间接引用引发的，元组仍然属于不可变数据类型。

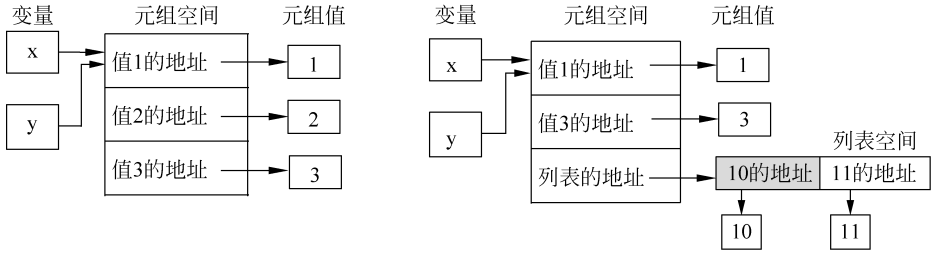

图 3-2　元组的共享引用

3. 列表、字典、集合的共享引用

列表、字典、集合的元素是不可变数据时，改变某个元素值时，数据存储空间的位置不变，数据值发生改变。

列表的共享引用示例：

```
>>> x = [1, 2, 3]
>>> y = x
>>> x, y
([1, 2, 3], [1, 2, 3])
>>> id(x), id(y)
(2349538144328, 2349538144328)
>>> x[1] = 100
>>> x, y
([1, 100, 3], [1, 100, 3])
>>> id(x), id(y)
(2349538144328, 2349538144328)
>>> x is y
True
```

字典的共享引用示例：

```
>>> i = {'F':1, 'H':2, 'Y':3}
>>> j = i
>>> i
{'Y': 3, 'H': 2, 'F': 1}
>>> j
{'Y': 3, 'H': 2, 'F': 1}
>>> id(i), id(j)
(2349538516360, 2349538516360)
>>> i['F'] = 99
>>> i
{'Y': 3, 'H': 2, 'F': 99}
>>> j
{'Y': 3, 'H': 2, 'F': 99}
>>> id(i), id(j)
(2349538516360, 2349538516360)
>>> i is j
True
```

集合的共享引用示例：

```
>>> m = {3, 4, 5}
>>> n = m
>>> m, n
({3, 4, 5}, {3, 4, 5})
>>> id(m), id(n)
(2349538945064, 2349538945064)
>>> m.add(99)
>>> m, n
({99, 3, 4, 5}, {99, 3, 4, 5})
>>> id(m), id(n)
(2349538945064, 2349538945064)
>>> m is n
True
```

列表、字典、集合的元素是可变数据时，进行赋值操作或调用相关数据类型的方法改变某个可变数据元素值时，值的变化就会各不相同。例如：

```
>>> x = [1, 2, [9, 10]]
>>> y = x
>>> id(x)
2349539132360
>>> id(y)
2349539132360
>>> x = x + [4]          # 发生了浅复制
>>> x
[1, 2, [9, 10], 4]
>>> y
[1, 2, [9, 10]]
>>> id(x)
2349539131656
>>> id(y)
2349539132360
>>> x[0] = 100           # 独立引用
>>> x
[100, 2, [9, 10], 4]
>>> y
[1, 2, [9, 10]]
>>> y[2][0] = -1         # 共享引用
>>> x
[100, 2, [-1, 10], 4]
>>> y
[1, 2, [-1, 10]]
>>> id(x)
2349539131656
>>> id(y)
2349539132360
```

以上代码段发生了浅复制，图 3-3 所示的是语句 x=x+[4]执行前的内存状态，当执行 x=x+[4]时，Python 分配一个新的存储空间用于存储新的列表，并将原来 x 列表中的数据地址复制到新空间中，语句 x[0]=100 改变了新空间的引用值。图 3-4 中的“间接列表空间”是由 x 和 y 共享的，语句 y[2][0]=-1 改变了“间接列表空间”中的引用值，因而在显示 x 时，其值发生了改变。语句 x=x+[4]浅复制后的内存状态如图 3-4 所示。

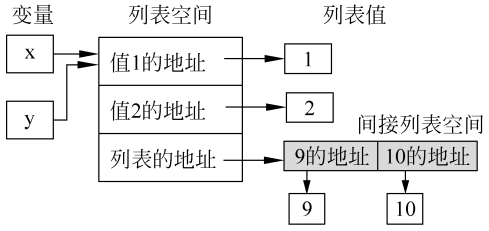

图 3-3　语句 x=x+[4]执行前的内存状态

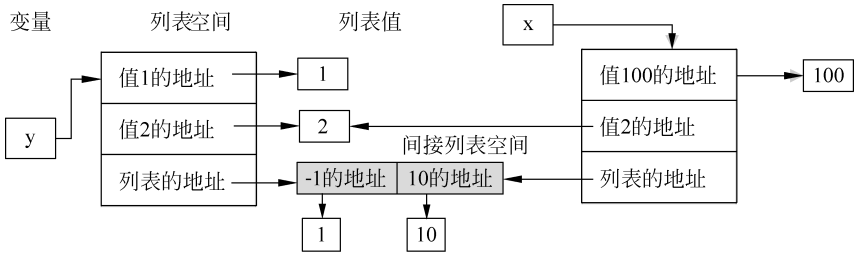

图 3-4　语句 x＝x＋[4]浅复制后的内存状态

下述代码也发生了浅复制：

```
>>> x = [[1, 2, 3]] * 3
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>> x[0][0] = 10
>>> x
[[10, 2, 3], [10, 2, 3], [10, 2, 3]]
```

执行语句 x＝[[1,2,3]] * 3 后，新列表的三个数据元素都指引到同一个间接列表空间，语句 x[0][0]＝10 貌似只改变了一个间接值，但结果却是引发了连锁反应。

与浅复制相对应的是深复制，例如有如下代码：

```
>>> import copy
>>> names = ["小明", "小红", "小黑", ["粉色"], "小黄", "小白"]
>>> deep_names = copy.deepcopy(names)
>>> names[3][0] = "Pink"
>>> names
['小明', '小红', '小黑', ['Pink'], '小黄', '小白']
>>> deep_names
['小明', '小红', '小黑', ['粉色'], '小黄', '小白']
```

上述代码中调用 copy 模块中的 deepcopy()函数将发生深复制，使两个变量的值各自独立，其中的细节，请读者另行查阅资料或自行研究。

3.2.3　对象的垃圾回收机制*

Python 中万物皆对象，在前面已介绍的内容中，每个数据都是一个对象。后面在第 11 章还将专门介绍关于对象的知识。在赋值操作中，Python 会建立变量和对象之间的引用关系(如图 2-1 所示)。程序中需要使用数据或对象时，就会在内存中开辟一块内存空间存放该数据或对象。

1. Python 对象的存储问题

- 对于整数和短小的字符串等，Python 会执行缓存机制，即将这些对象进行缓存，不会为相同的对象分配多个内存空间。
- 容器对象(如列表、元组、字典等)存储的其他对象，仅仅是其他对象的引用，即记录了对象的地址，并不是这些对象本身。例如：

```
>>> import sys
>>> sys.getrefcount(3)        #3 在 Python 内部已经被多次引用,故引用计数大于 2
184
>>> h = 3
>>> x = [3, 3, 3]
>>> sys.getrefcount(3)
188
>>> id(h), id(x[0]), id(x[1]), id(x[2])
(1900216848, 1900216848, 1900216848, 1900216848)
>>> sys.getrefcount('OK')
21
>>> sys.getrefcount(123)
4
>>> sys.getrefcount(3.1)
2
```

上述代码中的 3、'OK'、123 属于整数和短小字符,Python 执行缓存机制。实数 3.1 被 getrefcount()查看引用计数时,3.1 作为参数传递给了 getrefcount(),参数实际上创建了一个临时的引用。因此,getrefcount()所得到的结果,会比期望的多 1。

2. 关于引用计数器

在 Python 中,每个对象都有指向该对象的引用总数,即引用计数。每增加一个引用,引用计数增加 1;每减少一个引用,引用计数减少 1。

会增加引用计数的几种情况有:

- 对象被创建,计数增加 1;
- 对象被引用,计数增加 1;
- 对象被当作参数传入函数,计数增加 2,因为调用函数中有两个属性在引用该对象;
- 对象存储到容器对象中,计数增加 1。

会减少引用计数的几种情况有:

- 对象的别名被销毁,计数减少 1;
- 对象的别名被赋予其他对象,计数减少 1;
- 对象离开自己的作用域,计数减少 1;
- 对象从容器对象中删除,或者容器对象被销毁,计数减少 1。

使用 sys 模块中的 getrefcount()函数可以查看对象的引用计数。例如:

```
#1.   import sys
#2.   class Person(object):
#3.         pass
#4.   p = Person()
#5.   p1 = p
#6.   print(sys.getrefcount(p))
#7.   p2 = p1
#8.   print(sys.getrefcount(p))
#9.   p3 = p2
#10.  print(sys.getrefcount(p))
#11.  del p1
#12.  print(sys.getrefcount(p))
```

利用引用计数器方法，在检测到对象引用个数为 0 时，对普通的对象进行释放内存的机制处理（该机制如何处理，本书不展开讨论）。

3. 垃圾回收

当 Python 中的对象越来越多时，它们将占据越来越大的内存。Python 会在适当的时候，启动垃圾回收（garbage collection），将没用的对象清除。

垃圾回收时，Python 不能执行其他的任务。频繁的垃圾回收将大大降低 Python 的工作效率。如果内存中的对象不多，就没有必要总启动垃圾回收。所以，Python 只会在特定条件下，自动启动垃圾回收。

3.3　数据的输入/输出

3.3.1　标准输入/输出

被程序处理的数据可以从键盘输入，也可以从文件读入。程序的处理结果则可以显示在屏幕上，或存入文件。所谓的标准输入/输出是指从键盘输入和在屏幕显示，又叫控制台输入/输出。

1. 标准输入

Python 中的 input()函数用于实现标准输入，其格式为：

```
input(prompt = None)
```

参数 prompt 是输入时的提示文字，属于可选项。input()函数将接收标准输入设备键盘输入的一行信息，并返回一个字符串。返回的字符串是去除行末回车符后的输入内容。例如：

```
>>> input('Please input your name:')
Please input your name:Sam↙
'Sam'
```

上述示例中，第 2 行前面的文字是 input()函数执行时的提示，Sam 是运行时敲击键盘输入的内容。第 3 行是函数的返回值，两个单引号表示返回值是一个字符串。

如果输入的内容是整数或浮点数，则需要使用 int()或 float()函数进行转换。例如：

```
#1.  >>> input('Please input your age:')
#2.  Please input your age:19↙
#3.  '19'
#4.  >>> int(input('Please input your age:'))
#5.  Please input your age:19↙
#6.  19
#7.  >>> float(input('Please input your score:'))
#8.  Please input your score:75.5↙
#9.  75.5
```

上述示例中#6 行和#9 行显示的返回值中无单引号，即其返回值分别为整数类型和浮

点数类型。

2. 多数据同时输入并转换

单独使用 int()或 float()函数只能输入一个数据并进行数据类型转换,若想要一次输入多个数据并转换成整型或浮点数,则可以用以下几种方法。

1) 利用 eval()函数实现多数据同时输入

eval()函数可以计算参数字符串中的表达式或通过 compile()执行一个代码对象,其格式为:

```
eval(source, globals = None, locals = None)
```

参数 source 是一串待计算的表达式或待执行的代码。例如:

```
>>> eval('help(eval)')
Help on built - in function eval in module builtins:

eval(source, globals = None, locals = None, /)
    Evaluate the given source in the context of globals and locals.

    The source may be a string representing a Python expression
    or a code object as returned by compile().
    The globals must be a dictionary and locals can be any mapping,
    defaulting to the current globals and locals.
    If only globals is given, locals defaults to it.
```

通过函数 eval()执行字符串'help(eval)'中的 help(eval)函数,help()函数返回 eval()函数的帮助信息。又如:

```
>>> a = 100
>>> b = eval('a / 2.0')
>>> b
50.0
```

上述代码中,b=eval('a/2.0')等价于 b=a/2.0,所以 b 的值变为 50.0。

再比如,可用如下代码输入多个数据到多个变量:

```
#1.  >>> a, b, c = eval(input('a,b,c = '))
#2.  a,b,c = 1,2,3↙
#3.  >>> a
#4.  1
#5.  >>> b
#6.  2
#7.  >>> c
#8.  3
```

上述代码中,#2 行中输入的 1,2,3 使得 #1 行的代码等价于 a,b,c=1,2,3。

2）利用字符串的切片和 map()函数实现多数据同时输入

字符串的切片格式：

```
S.split(sep = None, maxsplit = -1) -> list of strings
```

S 是被处理的字符串，参数 sep 是切片字符，maxsplit 是切片的次数。函数返回一个包含切片后子串的列表。若省略 sep，则切片字符是空白字符（包括空格、Tab 和回车符\n），若省略 maxsplit，则不限制切片次数，即遇到参数 sep 指定的字符都切片。例如：

```
>>> '1 2 3'.split(' ',1)        # '1 2 3'的 1、2、3 之间有个空格，共切片 1 次
['1', '2 3']
>>> '1 2\t3\n4 5'.split()       # 字符串的 1、2 之间和 4、5 之间有个空格
['1', '2', '3', '4', '5']
>>> '1,2,3,4,5'.split(',')
['1', '2', '3', '4', '5']
>>> input('Please input 5 number:').split()
Please input 5 number:1 2 3 4 5 ↙
['1', '2', '3', '4', '5']
```

从上述示例代码中，可以观察到通过字符串切片函数可以将同时输入的整数或浮点数分离出来，但是分离出来的内容还是字符串，而不直接是整数或浮点数。int()或 float()函数只能转换一个数，如果要将多个字符串转换成同种数据类型结果则可以使用 map()函数。

map()可以让单参数的函数作用到序列或可迭代对象上，返回一系列的处理结果。其格式为：

```
map(func, * iterables)
```

调用 map()函数时，func 参数应设为单参数函数的函数名，参数 * iterables 为序列或可迭代对象，作用是将每个元素传递给单参数处理，得到各项函数值，并以 map 对象返回。例如：

```
>>> x = input('Please input 5 number:')
Please input 5 number:1;2;3;4;5 ↙
>>> y = x.split(';')
>>> y
['1', '2', '3', '4', '5']
>>> map(int, y)
<map object at 0x000001DE8A273390>
>>> z = map(int, y)
>>> z     # 返回的 map 对象是可迭代对象，但不能直接显示其内部各元素值
<map object at 0x000001DE8A273438>
>>> w = list(z)
>>> w
[1, 2, 3, 4, 5]
>>> w = list(map(int, input('Please input 5 number:').split(';')))
Please input 5 number:1;2;3;4;5 ↙
>>> w
[1, 2, 3, 4, 5]
```

3. 标准输出

程序执行中产生的处理结果,需要以一定方式展示出来,其中最常用的方式是显示在屏幕上。Python 中的标准输出函数是 print(),其格式为:

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

其作用为显示参数 value 到输出流或标准输出设备(即屏幕)sys.stdout。参数 file 用于指定输出流,若缺省 file 则输出到屏幕。参数 sep 指定多输出项之间的间隔字符,若省略 sep 则以空格间隔。参数 end 指定显示最后一项 value 后显示的字符,若省略 end 则显示 \n,即默认情况,显示数据后会换行。例如:

```
>>> print(1, 'OK', 98.12, [1, 2, 3], None)
1 OK 98.12 [1, 2, 3] None
>>> print(1, 'OK', 98.12, [1, 2, 3], None, sep = '!'); print('----')
1!OK!98.12![1, 2, 3]!None
----
>>> print(1, 'OK', 98.12, [1, 2, 3], None, sep = '!', end = ''); print('----')
1!OK!98.12![1, 2, 3]!None ----
```

3.3.2 格式化输出

很多应用中,对输出内容是有格式要求的。例如,很多实验数据需要保留指定位数的小数,而使用 print()函数直接输出时,小数位数是 Python 内部自动控制的。例如:

```
>>> a = 19 / 7
>>> a
2.7142857142857144
```

Python 中,可以用以下方法控制输出内容的格式:

- 利用字符串格式化运算符%;
- 利用内置函数 format();
- 利用字符串的 format()方法。

1. 字符串格式化运算符%

这是 Python 的早期版本提供的一种输出格式化方法。字符串格式化运算符%的使用格式:

```
格式字符串 % (数据项 1, [数据项 2, [数据项 3, …]])
```

格式字符串可以由普通字符和格式字符组成,普通字符按原样输出,而一组格式字符与一个数据项对应,由以下内容组成:

```
% [-] [+] [0] [m] [.n] 数据类型说明符
```

格式字符串由%开始,数据类型说明符则根据数据项的数据类型来指定,如表 3-1

所示。

表 3-1　格式字符串中不同符号的含义

格 式 符 号	格式化结果
%%	字符百分号%
%c	单个字符
%s	字符串,等价于 str()的返回值
%r	字符串,等价于 repr()的返回值
%d 或%i	十进制整数
%o	八进制整数
%x 或%X	十六进制整数,其中的字符用小写或大写
%e 或%E	科学计数法表示的浮点数,用 e 或 E 表示
%f 或%F	非科学计数法表示的浮点数
%g 或%G	浮点数,系统自动根据值的大小采用%e、%E、%f 或%F
−	左对齐输出
+	对正数加正号
0	空位用 0 填充
m	m 是数字,指定最小宽度
.n	n 是数字,指定精度,采用%e、%E、%f、%F、%g 或%G 时含义不同

下面的示例展示字符串格式化运算符%的用法。

```
>>> print('%%\t%c\t%s\t%r' % ('A', 'abc', 'abc'))
%%   A   abc   'abc'
```

这里的%%将输出一个%,后面紧跟着一个\t,表示跳过一个制表位,接着%c 对应第一个输出项'A',以单个字符形式显示为 A,后面又是一个\t,继续跳过一个制表位,接着%s 对应的是第二个输出项'abc',以%s 格式化字符串时不显示字符串的定界符,然后又是跳过一个制表位后,%r 对应了第三个输出项'abc',用%r 格式化字符串时会显示字符串的定界符。

```
>>> print('%d,%i,%o,%x,%X' % (299, 299, 299, 299, 299))
299,299,453,12b,12B
```

这个示例中 5 个输出项分别对应了 299 的十进制、十进制、八进制、十六进制的小写形式和十六进制的大写形式。

```
>>> print("Name: %s Age: %d Height: %f" %("Aviad", 25, 1.83))
Name: Aviad Age: 25 Height: 1.830000
```

上面 3 个输出项都是根据各自的数据类型采用了 Python 默认的输出格式。

```
>>> print("Name: %10s Age: %8d Height: %8.2f" % ("Aviad", 25, 1.83))
Name:      Aviad Age:       25 Height:       1.83
```

这里的输出项都指定了宽度，当指定的宽度大于数据长度时，将在前面补空格填满指定宽度。输出 1.83 时，还规定了小数点后保留 2 位，即%8.2f 表示宽度为 8 位，精度为 2 位。

```
>>> print("Name: %-10s Age: %-8d Height: %-8.2f" % ("Aviad", 25, 1.83))
Name: Aviad      Age: 25       Height: 1.83
```

格式字符串中的－代表左对齐，当指定宽度时，补的空格是在数据的右边。

```
>>> print("Name: %010s Age: %08d Height: %08.2f" % ("Aviad", 25, 1.83))
Name:     Aviad Age: 00000025 Height: 00001.83
```

格式字符串中含有 0，表示填充字符是 0，而不是空格，0 仅对数值类型有效。

```
>>> print("Name: %010s Age: %0+8d Height: %+08.2f" % ("Aviad", 25, 1.83))
Name:     Aviad Age: +0000025 Height: +0001.83
```

格式字符串中含有＋，表示正数前面要有＋。

```
>>> print("Name: %(name)010s Age: %(age)0+8d Height: %(height)+08.2f" % \
       {'name':"Aviad", 'height':1.83, 'age':25})     #输出字典数据时的用法
Name:     Aviad Age: +0000025 Height: +0001.83
```

格式字符串中出现在括号内的是字典的键，输出时会在括号处替换为与键对应的值。

```
>>> '%g' % 12345.456789901234          #自动根据数据大小选择小数形式或指数形式
'12345.5'
>>> '%f' % 12345.456789901234          #小数形式
'12345.456790'
>>> '%e' % 12345.456789901234          #指数形式
'1.234546e+04'
>>> '%14.5g' % 12345.456789901234      #自动形式中的精度指有效数字数
'12345'
>>> '%14.5f' % 12345.456789901234      #小数形式中的精度指小数点后的位数
'12345.45679'
>>> '%14.5e' % 12345.456789901234      #指数形式中的精度指前面数字的小数位数
'1.23455e+04'
>>> '%-+0*.*f' % (16, 2, 100.93)       #在运算符%后指定最小宽度和精度
'+100.93 '
```

上面最后一个示例比较特殊，里面出现了 *.*，这两个 * 分别对应的是括号中的前两项 16 和 2，相当于是 16.2，所以虽然括号中有三个数字，但输出项只有最后的 100.93。

2. 内置函数 format()

format()函数用于将单项数据格式化，其格式为：

```
format(输出项[, 格式字符串])
```

当省略第二个参数时，format 函数等价于 str()，即将输出项转换为字符串。格式字符

串中的基本格式控制符如下：

- d、b、o、x、X 分别用十进制、二进制、八进制和十六进制输出整数。例如：

```
>>> print(format(95, 'X'), format(95, 'o'), format(95, 'b'))
5F  137  1011111
```

- f 或 F、e 或 E、g 或 G 分别用小数形式、科学计数和自动判定来输出浮点数。例如：

```
>>> print(format(162.28193, 'e'), format(162.28193, 'g'), format(162.28193, 'f'))
1.622819e+02  162.282  162.281930
```

- c 输出字符，字符的 ASCII 码由参数输出项指定。
- %输出百分数，数值由输出项指定。
- 输出浮点数时，带千分位符，。例如：

```
>>> print(format(31416.123, ',f'))
31,416.123000
```

注意，小数点后 123 与 000 之间是没有千分位符的，因为这里的 000 不是小数点后的有效位数，只是将空白区填充满 0，真正的有效小数位数只有 3 位。

- 用形如 m.n 的格式来控制输出宽度和精度，m 和 n 都是数字。
- 输出整数或浮点数时，可以使用＋表示正数带正号。
- 在指定输出宽度时，在输出宽度前可以使用 0、＜、＞、^表示用 0 填充空位(默认用空格填充空位)、左对齐、右对齐和居中对齐。

以上格式控制符根据数据类型的不同，可以部分同时使用。例如：

```
>>> print(format(2.11, '10'), format(2.11, '010'), format(2.11, '+10'))
      2.11 0000002.11     +2.11
```

这个示例中输出了三次 2.11，它们的宽度都是 10，第一个是左边补空格，第二个是左边补 0，第三个是 2.11 前要出现＋。

```
>>> print(format('aaa', '<10'), '|', format('aaa', '^10'), '|', format('aaa', '>10'), '|')
aaa       |   aaa    |       aaa|
```

这个示例是左对齐、居中和右对齐的示例。

```
>>> print(format(3.1416, '8.3f'), '|', format(3.1416, '08.3f'), '|', format(3.1416, '+08.3f'))
    3.142 | 0003.142 | +003.142
>>> print(format(3.1416, '<8.3f'), '|', format(3.1416, '<08.3f'), '|', format(3.1416, '<+08.3f'))
3.142   | 3.142000 | +3.14200
```

上面两个示例都是浮点数的输出示例，可以控制宽度和精度，也可以控制对齐方式。

3. 字符串的 format()方法

Python 中的字符串类型有一个 format()方法，利用该方法可以格式化字符串。字符串

format()方法的调用格式为：

```
格式字符串.format([键名 0 = ]输出项 0, [键名 1 = ]输出项 1, [键名 2 = ]输出项 2,…)
```

格式字符串中可以包括普通字符和格式说明模板，可以有多个格式说明模板，普通字符原样输出。格式说明模板的格式为：

```
{[输出项序号|键名][:格式说明符]}
```

其中，{}是输出模板的定界符，输出项序号为 0、1、2、…，分别对应输出项 0、输出项 1、输出项 2…。格式说明模板中的键名与输出项前的键名匹配。省略输出项序号和键名时，多个格式说明模板与多个输出项按自然位置对应显示。除输出项序号以外的格式说明符与内置函数 format()中的格式说明符含义基本一致。例如：

```
>>> print('Name: {0} Age: {2} Height: {1}'.format("Aviad", 1.83, 25))
Name: Aviad Age: 25 Height: 1.83
```

这个示例中{}内的是序号，0 对应第一个输出项"Aviad"，1 对应第二个输出项 1.83，2 对应第三个输出项 25。

```
>>> print('Name: {name} Age: {height} Height: {age}'.format(name = "Aviad",
      height = 1.83, age = 25))
Name: Aviad Age: 1.83 Height: 25
```

这个示例中{}内的是键名，name 对应"Aviad"，height 对应 1.83，age 对应 25。

```
>>> print('{0:010b}|{0:>10o}|{0:^10x}|{0:<10X}'.format(95))
0001011111|       137|   5f   |5F
>>> print( '{0:018}|{0:>18}|{0:^18.2}|{0:<18.3}'.format(3.14159))
000000000003.14159|           3.14159|       3.1        |3.14
```

这两个例子都是同时规定序号和格式的例子。在“:”前面的是序号，本例中都是 0，对应的是第一个输出项；在“:”后面的是格式控制。

```
>>> print('{:14}|{:>14}|{:<14}'.format('test', 'test', 'test'))
test          |          test|test
```

“:”前面的序号可以省略，当省略序号时，自动根据格式说明模板的顺序，与输出项按照自然顺序相对应。

```
>>> print( '{0:*>18}|{0:?^18}|{0:-<18}'.format('test'))
**************test|???????test???????|test----------------
```

字符串类型的 format()方法对于填充字符，若输出项是整数或浮点数，填充字符只能是 0 或空格，默认为空格；若输出项是字符串则可以指定任意填充字符。

3.4　顺序结构程序举例

1. 程序编写的一般方法

看到一个问题，到最终编写出正确的程序来处理这个问题，一般需要经历以下步骤：①分析问题；②确定算法；③编写代码；④测试代码；⑤提交并发布代码。

大多数程序的基本功能框架如下：

- 获取原始数据；
- 对数据进行处理；
- 输出或存储处理结果。

2. 顺序结构程序举例

程序的主结构是顺序结构，即程序总是按照语句出现的先后顺序依次执行的。以下是一些顺序结构的示例，通过它们可以对顺序结构、编程的一般步骤和程序基本功能框架有一定的了解。

【例 3-2】 A 汽车从甲地开往乙地，以平均速度 45km/h 行驶，B 汽车从乙地开往甲地，以平均速度 53km/h 行驶，两辆车行驶了 2 小时 13 分钟后相遇。编写程序，输入 A、B 的速度和行驶的时间，求甲乙两地相距多少公里。

以下是本例按编程的一般步骤示范的解题过程。本书后续的其他示例一般只给出问题的解题思路和程序代码。

（1）分析问题：这是个数学问题，找到基本的数学公式“距离＝（A 速度＋B 速度）＊行驶时间”就能求解。

（2）确定算法：算法流程如图 3-5 所示，按程序三部曲（输入、处理、输出）结合本问题的特点，确定算法的每一步。

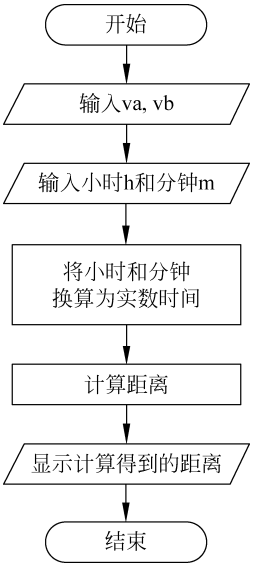

图 3-5　例 3-2 的流程图

（3）编写代码：根据算法，将每个步骤落实为具体的代码。本题的程序代码如下：

```
#1.  va, vb = eval(input('输入 A 车和 B 车的速度:'))
#2.  h, m = eval(input('输入行驶时间:'))
#3.  time = h + m / 60
#4.  s = time * (va + vb)
#5.  print('甲乙两地相距{}公里。'.format(s))
```

（4）测试代码：在 IDLE 中，输入上述代码，并参考本书第 12 章中的介绍，先改正语法错误、运行时错误后，输入若干测试数据，观察每次运行结果的正确性。以下是某次测试的结果。

```
输入 A 车和 B 车的速度: 45,53↙
输入行驶时间: 2,13↙
甲乙两地相距 217.23333333333335 公里。
```

如果发现运行结果不正确则程序必然存在逻辑错误，需要逐行分析代码或采用 12.4 节

的调试工具来发现错误点，并改正。若所有测试数据都有正确的运行结果，则可以进行最后一步——提交并发布代码。

【例 3-3】 解析几何中，求点(x,y)到直线$Ax+By+C=0$的距离公式是$\frac{|Ax+By+C|}{\sqrt{A^2+B^2}}$，编写程序输入点的坐标$(x,y)$值和直线方程的参数$A$、$B$、$C$的值，求点到直线的距离。

解题过程：距离公式中涉及求绝对值、求平方和求开平方，可以使用内置abs()函数求绝对值，平方就是一个数乘以自身，开平方可以用乘方运算**求解。

程序代码如下：

```
#1.  x, y = eval(input('x,y:'))
#2.  a, b, c = eval(input('方程系数 A,B,C:'))
#3.  s = abs(a * x + b * y + c) / (a * a + b * b) ** 2
#4.  print('点到直线的距离：{}'.format(s))
```

以下是某次测试结果：

```
x,y:0,1↙
方程系数 A,B,C:2,3,4↙
点到直线的距离：0.04142011834319527
```

【例 3-4】 从键盘输入一个三位整数，计算该数中各位数字之和。例如，输入392，各位数字之和是3+9+2=14。

解题思路：本题的难点是如何将整数中的各位数字提取出来，以下是三种提取数字的方法。

方法一：

```
#1.  x = input('请输入一个三位数：')
#2.  x = int(x)
#3.  a = x // 100                #获取百位数
#4.  b = x // 10 % 10            #获取十位数
#5.  c = x % 10                  #获取个位数
#6.  print(a + b + c)
```

方法二：

```
#1.  x = input('请输入一个三位数：')
#2.  x = int(x)
#3.  a, b = divmod(x, 100)       #a 得到百位数,b 得到后两位数
#4.  b, c = divmod(b, 10)        #b 得到十位数,c 得到个位数
#5.  print(a + b + c)
```

方法三：

```
#1.  x = input('请输入一个三位数：')
#2.  a, b, c = map(int, x)       #采用 map()函数依次得到百、十、个位数
#3.  print(a + b + c)
```

这三种方法都可以得到同样的结果。下面是某次的测试结果：

```
请输入一个三位数：392↙
14
```

【例 3-5】 从键盘输入一个三位整数，将该整数转换为英文表达。例如，输入 392，输出 three hundred and ninety two。

有人设计了如下代码：

```
#1.  x = input('请输入一个三位数:')
#2.  a, b, c = map(int, x)
#3.  eng1 = ['','one','two','three','four','five','six','seven','eight','nine']
#4.  eng2 = ['','ten','twenty','thirty','forty','fifty','sixty','seventy','eighty','ninety']
#5.  print('{} hundred and {} - {}'.format(eng1[a],eng2[b],eng1[c]))
```

上述代码的解题思路：列表 eng1 中存放单个数字对应的英文单词，eng2 中存放 10、20 等整十数的英文单词，a、b、c 是分离出来的各位数字，用 a、b、c 作为列表下标去获取数字对应的英文单词。

运行程序，输入如下数据测试代码：

```
请输入一个三位数：392↙
three hundred and ninety - two
```

以上数据测试正确。但是，实际上本题的解法是有缺陷的，当末两位是 11～19 或末尾为 0 时翻译结果是不正确的，要解决这个问题就需要用到选择结构。以下是错误的运行结果：

```
请输入一个三位数：312↙
three hundred and ten - two
```

3.5　习　　题

1. 简述 Python 程序中注释的格式。

2. 简述 Python 程序中缩进的作用。

3. Python 的语句是如何将多条语句写在一行上的？Python 对将一条语句写到多行上有哪些规定？

4. Python 有哪几种赋值语句？简述各自的格式。

5. 简述 Python 是如何实现输入和输出的。

6. 求 x 的值：

(1) 已知 x=3、y=2，执行表达式 x * =y+8 后 x 的值。

(2) 已知 x=10，执行表达式 x+=x 后 x 的值。

(3) 执行表达式 y,z,x=4,16,32 后，x 的值。

7. 运行以下程序时，输入“1,2,3”，写出运行结果。

```
#1.  i = input('Please input:')
#2.  a, b, c = eval(i)
#3.  m, n, r = i.split(',')
#4.  x, y, z = map(int, i.split(','))
#5.  print(a, b, c, m, n, r, x, y, z)
```

8. 运行以下程序时，输入“123”，写出运行结果。

```
#1.  x = input('Please input:')
#2.  i, j, k = map(int, x)
#3.  print(i, j, k)
```

9. 写出下列程序的运行结果。

(1)

```
#1.  print('Hi,world', end = '')
#2.  print("I'm...")
```

(2)

```
#1.  a = 'Hi,world'
#2.  b = "I'm ..."
#3.  print(a, b, a, b, a, b, sep = '!\n')
#4.  print(b)
#5.  print('too tied!')
```

(3)

```
#1.  a = '甲'
#2.  b = '乙'
#3.  c = '丙'
#4.  print('第一名：{},第二名：{},第三名：{}.'.format(a, b, c))
#5.  print('第一名：{2},第二名：{0},第三名：{1}.'.format(a, b, c))
#6.  print('第一名：{s2},第二名：{s1},第三名：{s3}.'.\
#7.    format(s1 = a, s2 = b, s3 = c))
```

10. 请计算出下列语句中各个赋值运算符左边的变量的值。注意，并不是按顺序执行这些语句，假定在每条前都已安排下列语句：

```
#1.  i = 3;
#2.  j = 5;
#3.  x = 4.3;
#4.  y = 58.209;
```

(1) k=j*i;

(2) k=j/i;

(3) z＝x/i；

(4) k＝x//i；

(5) z＝y/x；

(6) k＝y//x；

(7) i＝3＋2＊j；

(8) k＝j%i；

(9) k＝j%i＊4；

(10) i＋＝j；

(11) j－＝x；

(12) i%＝j。

11. 编写程序，实现从键盘输入学生的平时成绩、期中成绩、期末成绩，计算学生的学期总成绩。学生的学期总成绩＝平时成绩＊15%＋期中成绩＊25%＋期末成绩＊60%。

12. 编写程序，从键盘输入一个三位整数，计算该数的逆序数。例如，输入 392，输出 293。

13. 对于一元二次方程 $ax^2+bx+c=0$，输入其三个系数 a、b、c，输出方程的根。注：Python 有复数类型，且可以用 sqrt()直接求得复数结果。

14. 有 4 个进水管(A、B、C、D)可以往水箱里注水，如果单开 A，则 a 小时注满；如果单开 B，则 b 小时可以注满；如果单开 C，则 c 小时注满；如果单开 D，则 d 小时可以注满。编写程序，输入 a、b、c、d 的值，计算 4 个水管同时注水，注满水箱需要多少小时。