第3章 分支程序设计

本章学习目标

- 理解条件表达式、程序异常的概念
- 掌握关系、逻辑运算符及表达式
- 熟练掌握使用 if-else 语句、if 语句设计简单分支程序的方法
- 熟悉 os 库、math 库的常用函数及用法
- 理解闪屏问题及其处理的两种常用方法
- 熟练掌握使用分支嵌套和 if-elif-else 语句设计复杂分支程序的方法
- 掌握程序异常的处理方法

本章研究分支结构的程序设计。主要介绍关系与逻辑运算、双路分支语句 if-else、条件表达式、单路分支语句 if、os 库、math 库、分支嵌套、多路分支语句 if-elif-else、程序异常及其处理。

3.1 关系与逻辑运算


扫一扫

3.1.1 关系运算

1. 运算符和表达式

关系运算也叫作比较运算。关系运算用来确定两个对象之间的大小关系。关系运算的结果是逻辑值 True 或 False。Python 提供了 6 个关系运算符，分别是大于(>)、小于(<)、大于或等于(>=)、小于或等于(<=)、等于(==)和不等于(!=)，如表 3-1 所示。

表 2-1　关系运算符和表达式

运　算　符	实 施 运 算	优　先　级	表　达　式
＞ ＜ ＞＝ ＜＝	大于 小于 大于或等于 小于或等于	高	$x > y$ $x < y$ $x >= y$ $x <= y$
＝＝ ！＝	等于 不等于	低	$x == y$ $x != y$

2.4 点说明

(1) 关系运算的适用范围。

不是所有类型的数据之间都可进行关系运算。数值型与字符串类型之间、复数与复数之间不可以进行表 2-1 所示的前 4 种运算。

不同数据类型数据间关系运算的程序示例如图 3-1 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> 1 >= -10     #整数间比较，合法
True
>>> 1.5 <= 2     #小数与整数间比较，合法
True
>>> 1.5 > 1.25   #小数与小数间比较，合法
True
>>> "ab" > "Ab" #字符串与字符串间比较，合法
True
>>> True >= False   #逻辑值间比较，合法
True
>>> 1 >= "1"     #整数与串间比较，非法
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    1 >= "1"     #整数与串间比较，非法
TypeError: '>=' not supported between instances of 'int' and 'str'
>>> 2j > -1j  #复数之间比较非法
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    2j > -1j  #复数之间比较非法
TypeError: '>' not supported between instances of 'complex' and 'complex'
>>>
```

图 3-1　不同数据类型数据间关系运算的程序示例

在这个程序中，第 1 条语句是整数 1 与整数－10 进行比较，结果是 True。第 2 条语句是浮点数 1.5 与整数 2 进行比较，结果是 True。第 3 条语句是浮点数 1.5 与浮点数 1.25 进行比较，结果是 True。第 4 条语句是字符串“ab”与字符串“Ab”进行比较，结果是 True。第 5 条语句是逻辑值 True 与 False 进行比较，2.2.3 节中介绍过，逻辑值 True 与 False 可以看作整数 1 和 0，可以比较大小，结果是 True。第 6 条语、第 7 条语句分别是整数 1 和字符串“1”、复数 2j 与复数-1j 之间进行比较，程序运行出错，系统提示整数与字符串之间、复数与复数之间不支持“＞＝”和“＞”运算。

(2) 字符串之间的关系运算。

字符串之间比较大小时，是自左向右逐个比较对应位置字符的 Unicode 码，只要能确定结果，就停止比较。

① 对于字母来说，小写字母 a 的 Unicode 码是 97，大写字母 A 是 65，大写字母之间、小写字母之间均连续编码，小写字母比与其对应的大写字母的 Unicode 码大 32。

② 对于数字字符来说，字符 0 的 Unicode 码是 48，字符 0～9 也是连续编码。

③ 对于汉字来说，可以理解为是对汉字对应的汉语拼音之间进行比较。

字符串之间关系运算的程序示例如图 3-2 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> #字符串间比较
>>> "Abc">"a"      #"A"与"a"比较即确定结果为假
False
>>>
>>> "Abc"<="Abd"   #"c"与"d"比较确定结果为真
True
>>>
>>> #数字串比较
>>> "123" <= "11111"   #"2"与"1"比较确定结果为假
False
>>>
>>> #汉字间比较
>>> "马大强" > "马小强"   #"d"与"x"比较确定结果为假
False
>>>
```

图 3-2　字符串间关系运算的程序示例

在这个程序中，第 1 条语句是字符串“Abc”与“a”比较，因字母 A 小于 a 的 Unicode 码，所以结果为 False。第 2 条语句是字符串“Abc”与“Abd”比较，因为字母 c 小于 d 的 Unicode 码，所以结果为 True。第 3 条语句是“123”与“11111”比较，因为数字字符 2 大于 1 的 Unicode 码，所以结果为 False。第 4 条语句是汉字“大”与“小”比较，可以理解为是它们的汉语拼音“da”与“xiao”之间比较，因为字母 d 小于 x 的 Unicode 码，所以结果为 False。

(3) 参与关系运算的对象。

参与关系运算的对象可以是常量、变量和表达式。

不同对象之间关系运算的程序示例如图 3-3 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> #比较运算的对象
>>>
>>> 1==1.0   #常量间比较
True
>>>
>>> x=10;y=-5
>>> x>=y        #变量之间比较
True
>>>
>>> x+10>y**2   #表达式之间比较
False
>>>
>>> "123"*2 == "123123"   #表达式与常量间比较
True
>>>
>>>
```

图 3-3　不同对象间关系运算的程序示例

在这个程序中，第 1 条语句是两个常量 1 与 1.0 之间比较。第 2 条语句是两个变量 x 与 y 之间比较。第 3 条语句是两个表达式 $x+10$ 与 y**2 之间比较。第 4 条语句是一个表达式“123”* 2 和一个字符串常量“123123”之间比较。从运行结果可以看出，这些都是合法的。

(4) 比较运算的优先级。

在 6 种关系运算中，前 4 个的优先级相同，后 2 个的优先级相同。前 4 个高于后 2 个，但均低于数值运算和字符串运算。可以使用表达式 a<=x<=b 表示数学里的 x∈[a,b]。

不同优先级关系运算的程序示例如图 3-4 所示。

在这个程序中，第 1 条语句定义了变量 x 与 y，并分别为它们赋值 10 和 1.5。第 2 条语

```
File  Edit  Shell  Debug  Options  Window  Help
>>>  #关系运算的优先级
>>> x=10;y=1.5
>>> x+2>=y*10    #先求y*10得15,再求x+2得12,最后处理12>=15
False
>>>
>>> y>1==False   #先求y>1得Tue,再处理True==False
False
>>>
>>> "abc"[1:]!="bc"   #先处理"abc"[1:]得"bc",再处理"bc"!="bc"
False
>>>
>>> 10<=x<=15        #表示x∈[10,15]
True
>>>
>>> 2<y<3            #表示y∈(2,3)
False
>>>
```

图 3-4　不同优先级关系运算的程序示例

句是两个表达式之间比较，处理的顺序是先处理 $y*10$ 得 15，再处理 $x+2$ 得 12，最后处理 12>15，结果是 False。第 3 条语句是“>”和“==”两种运算连用，处理的顺序是先处理 $y>1$ 得 True，再处理 True==False，结果是 False。第 4 条语句是字符串切片运算与“!=”连用，处理的顺序是先进行切片得子串“bc”，然后处理“bc”!=“bc”，结果是 False。第 5 条语句是两个“<=”连用，表示 $x\in[10,15]$ 的条件，结果为 True。第 6 条语句是两个“<”连用，表示 $y\in(2,3)$ 的条件，结果为 False。

使用关系运算构造的几个典型表达式示例，如表 3-2 所示。

表 3-2　几个典型的关系表达式

关系表达式	结果为“真”时表示的条件
$x>0$	x 是正数
$x\%2==0$	x 是偶数
$x\%4==0$	x 能被 4 整除(x 是 4 的倍数)
$5<=x<=10$	$x\in[5,10]$
$5<x<10$	$x\in(5,10)$
"a"<=x<="z"	x 是小写字母
"A"<=x<="Z"	x 是大写字母
"0"<=x<="9"	x 是数字字符


扫一扫

3.1.2　逻辑运算

1. 概述

对于复杂的条件，只有关系运算是不够的，往往需要借助逻辑运算来构建。

逻辑运算是对逻辑值实施的运算。在 Python 中，任何类型的数据均可以作为逻辑值进行处理。

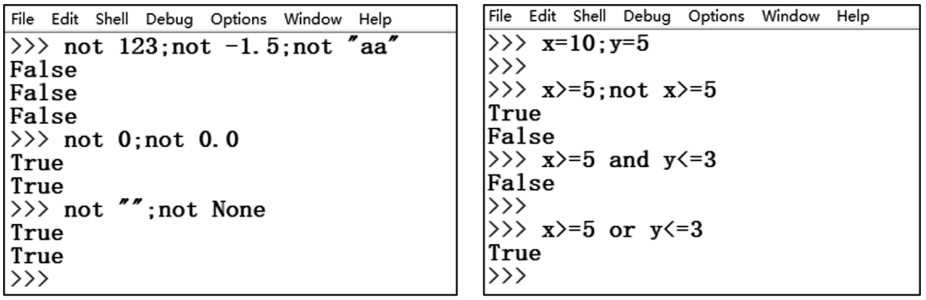
Python 规定：

- 任何非零数字或非空对象都为 True。

- 数字 0、空对象以及 None 都为 False。

None 是 Python 的一个关键字，用它表示空值。有关 None 的更多知识在 6.1.2 节介绍。

不同类型数据可以用作逻辑数据的程序示例参见图 3-5。



(a) 逻辑非运算　　(b) 3种逻辑运算

图 3-5　逻辑运算的程序示例

2. 运算符和表达式

Python 里提供了 3 种逻辑运算，如表 3-3 所示。not、and、or 是三个关键字。

表 3-3　逻辑运算符和表达式

运　算　符	实 施 运 算	优　先　级	表　达　式
not	逻辑非	高	not exp
and	逻辑与	低	exp1 and exp2
or	逻辑或	最低	exp1 or exp2

逻辑非(not)的优先级最高，逻辑与(and)次之，逻辑或(or)最低。

逻辑非(not)用在一个表达式的前面。逻辑与(and)、逻辑或(or)都用在两个表达式之间。

- 逻辑非(not)

  若该表达式的值为 True，结果就为 False，反之就为 True。
- 逻辑与(and)

  只有当两个表达式的值都为 True 时，结果才为 True，否则就为 False。
- 逻辑或(or)

  只有当两个表达式的值都为 False 时，结果才为 False，否则就为 True。

在 IDLE 命令交互方式下实施逻辑运算的程序示例如图 3-5 所示。

图 3-5(a)所示的是实施逻辑非(or)运算的程序示例，通过运行结果证明了前面介绍的任何非零数字或非空对象都为 True；数字 0、空对象以及 None 都是 False 的结论。图 3-5(b)所示的是 3 种逻辑运算的程序示例。第 1 条语句定义了变量 x 与 y，分别赋值 10 和 5。第 2 条语句里 $x>=5$ 的值为 True，not $x>=5$ 的结果为 False。第 3 条语句里 $x>=5$ 的值为 True，$y<=3$ 的值为 False，所以 $x>=5$ and $y<=3$ 的结果为 False。第 4 条语句里 $x>=5$

为 True，$y<=3$ 为 False，所以 $x>=5$ or $y<=3$ 的结果为 True。

使用逻辑运算构建的 5 个较复杂条件表达式的示例如表 3-4 所示。

表 3-4　5 个较复杂条件表达式的示例

表示的条件	表　达　式
x 是否为自然数(正整数)	type(x)==int and $x>0$
x 是否为字母	"a"<=x<="z" or "A"<=x<="Z"
x 是否满足 $x\in[5,10]$	$x>=5$ and $x<=10$ 等价于 $5<=x<=10$
a、b、c 是否构成三角形	$a+b>c$ and $b+c>a$ and $a+c>b$
y 是否闰年	(y%4==0 and y%100!=0) or y%400==0

3.1.3　is 运算符

1. 身份识别码

Python 为程序中的每个对象自动生成一个整数类型的身份识别码，用以对对象加以区分和识别。

通过 id 函数可以获取对象的身份识别码。该函数的调用格式如下。

```
id(对象)
```

这里的对象可以是常量、变量、表达式等。

获取对象识别码的程序示例如图 3-6 所示。

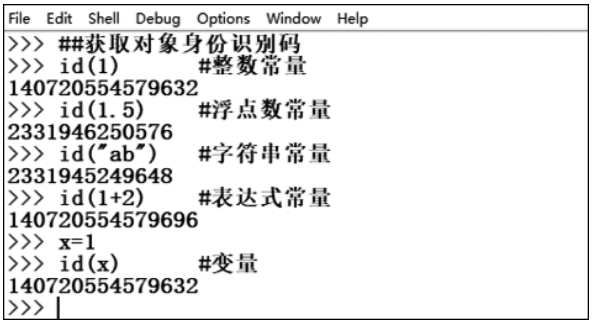


图 3-6　获取对象识别码的程序示例

在这个程序中，第 1 条语句获取了整数常量 1 的识别码。第 2 条语句获取了浮点数常量 1.5 的识别码。第 3 条语句获取了字符串常量"ab"的识别码。第 4 条语句获取了表达式 1+2 的识别码。第 5 条语句获取了变量 x 的识别码。这里请注意，变量 x 和常量 1 的识别码是一样的，因为 x 存的数据是 1。

2. is 运算符

is 是 Python 的一个关键字，用它可以判断两个对象的身份识别码是否一致。is 运算的使用方法如表 3-5 所示。

表 3-5　is 运算符和表达式

运　算　符	实 施 运 算	表　达　式
is	识别码一致	ob1 is ob2
is not	识别码不一致	ob1 is not ob2

其中，ob1 is ob2 等价于 id(ob1)==id(ob2)，若 ob1 和 ob2 的识别码相同，则为 True，否则就为 False。ob1 is not ob2 等价于 id(ob1)! =id(ob2)，若 ob1 和 ob2 的识别码不相同，则为 True，否则就为 False。

和 is 运算符不同的是，关系运算符“==”用来判断两个对象的值是否相等。

使用 is 和“==”运算符的程序示例如图 3-7 所示。

```
File  Edit  Shell  Debug  Options  Window  Help
>>> ##is与==运算的区别
>>>
>>> #==用于判断两个对象的值是否相等
>>> 1 == 1.0
True
>>> #is用于判断两个对象的识别码是否相同
>>> 1 is 1.0
False
>>>
```

图 3-7　使用 is 和“==”运算符的程序示例

在这个程序中，第 1 条语句用来判断整数 1 和浮点数 1.0 的值是否相等，结果是 True。第 2 条语句用来判断整数 1 和浮点数 1.0 的识别码是否相同，结果是 False。

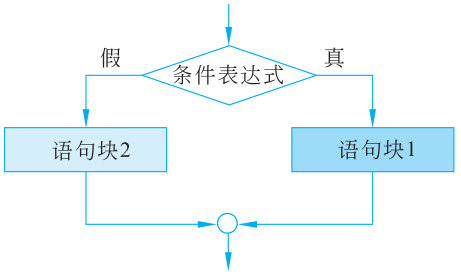
扫一扫

3.2　简单分支程序设计

3.2.1　双路分支语句 if-else

双路分支也叫作两路分支。双路分支是最常见的分支结构，它是根据某一条件的“真”或“假”，从两种情况中选择一个来执行。在 Python 中，双路分支使用 if-else 语句实现。if 与 else 是两个关键字。

双路分支的简单流程图和语句结构如图 3-8 所示。



语句结构

```
if 条件表达式：
    语句块1
else：
    语句块2
```

图 3-8　双路分支简单流程图与语句结构

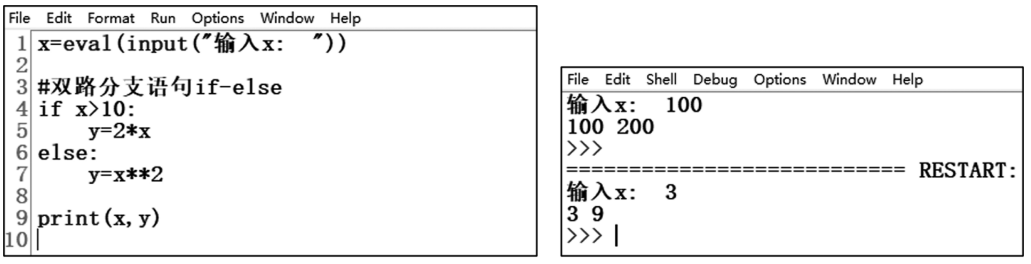
该结构的执行过程是：先处理 if 后面的条件表达式，如果结果为“真”(True)，则执行 if 后的语句块 1，结束整个结构；如果结果为“假”(False)，则执行 else 后的语句块 2，结束整个结构。

应该注意的两个问题如下。

(1) if 语句末尾与 else 后面的冒号是语法结构的一部分，不可以省略。

(2) 语句块 1 和语句块 2 要比 if 和 else 向右缩进一个 Tab 键位置。

双路分支程序示例如图 3-9 所示。



(a) 程序代码　　(b) 运行效果

图 3-9　双路分支程序示例

在这个程序中，第 1 行是为变量 x 输入数据的语句。第 4～7 行是一个双路分支，实现根据 x 的值求 y 的值。从运行结果可以看出，第 1 次执行时输入的数据是 100，满足 $x>10$ 的条件，所以执行了 if 后的 $y=2*x$，输出结果是 100 和 200。第 2 次执行时输入的数据是 3，不满足 $x>10$ 的条件，所以执行了 else 后的 $y=x**2$，输出的结果是 3 和 9。

3.2.2　条件表达式

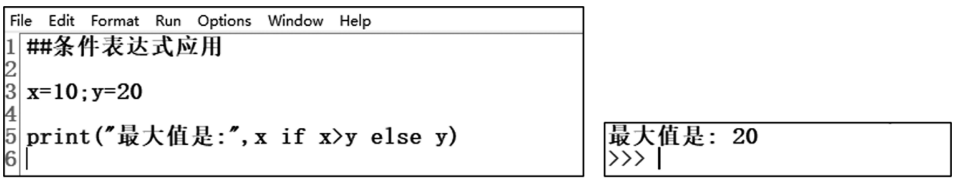
条件表达式是由 if 和 else 组成的表达式。

条件表达式有以下两种常用格式。

```
格式 1:表达式 1  if  条件表达式  else  表达式 2
```

处理过程是：先处理 if 后的条件表达式，如果结果为“真”(True)，则处理 if 前的表达式 1，并把它的值作为整个式子的值；如果结果为“假”(False)，则处理 else 后的表达式 2，并把它的值作为整个式子的值。

使用第 1 种格式的条件表达式程序示例如图 3-10 所示。



(a) 程序代码　　(b) 运行效果

图 3-10　使用第 1 种格式的条件表达式程序示例

在这个程序中，第 5 行把条件表达式直接作为 print 函数的一个参数。经过分析可知，该条件表达式的作用是求 x 与 y 的最大值，所以输出的结果是 20。

格式 2:语句 1　if　条件表达式　else　语句 2

处理过程是：先处理 if 后的条件表达式，如果结果为“真”(True)，则执行 if 前的语句 1，然后结束；如果结果为“假”(False)，则执行 else 后的语句 2，然后结束。

使用第 2 种格式的条件表达式程序示例如图 3-11 所示。

```
File  Edit  Format  Run  Options  Window  Help
1 #条件表达式应用
2 x=eval(input("输入x:  "))
3
4 print(x,2*x) if x>10 else print(x,x**2)
5
```

(a) 程序代码

```
File  Edit  Shell  Debug  Options  Window  Help
输入x:  100
100 200
>>>
========== RESTART:
输入x:  3
3 9
>>>
```

(b) 运行效果

图 3-11　使用第 2 种格式的条件表达式程序示例

在这个程序中，第 4 行把两条 print 语句分别放在了 if 的前面和 else 的后面。从运行结果可以看出，该程序实现的功能与图 3-9 所示的程序完全相同。比较两个程序的代码不难看出，引入条件表达式的程序，简化了代码，提高了运行效率。

3.2.3　单路分支语句 if

单路分支是双路分支的特例，它只有 if，没有 else。

单路分支的简单流程图和语句结构如图 3-12 所示。

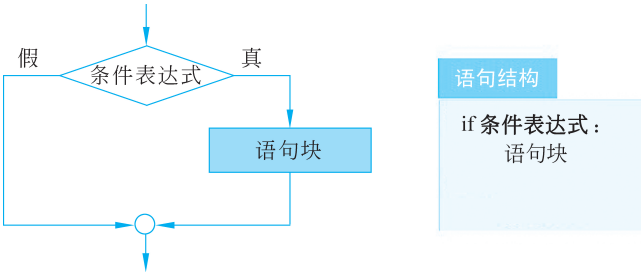


图 3-12　单路分支简单流程图与语句结构

该结构的执行过程是：先处理 if 后面的条件表达式，如果结果为“真”(True)，则执行 if 后的语句块，结束该结构；如果结果为“假”(False)，则跳过该结构。

单路分支程序示例如图 3-13 所示。

```
File  Edit  Format  Run  Options  Window  Help
1 #单路分支语句if
2 x=10
3 y=20
4
5 if x>=10:    #单路分支
6     y-=1
7
8 print(x,y)
9
```

(a) 程序代码

```
10 19
>>>
```

(b) 运行效果

图 3-13　单路分支程序示例

在上面的程序中，第 5～6 行是单路分支结构，因为 $x>=10$ 的条件成立，所以执行 if 后的 $y-=1$，输出的结果是 10 和 19。


扫一扫

3.2.4 2 个标准库模块

os 库和 math 库是 Python 两个重要的标准库模块，在实际编程中很常用。

1. os 库

os 是英文 operating system 的缩写。该模块提供了 Python 与操作系统之间的接口，通过它可以调用操作系统的一些功能。

本书只介绍 os 库的两个常用函数 system 和_exit，详细情况如表 3-6 所示。

表 3-6 os 库的两个常用函数及使用方法

函 数	实 现 功 能
system("cls")	清除屏幕上显示的内容
system("pause")	暂停程序执行，按任意键后继续执行
_exit(n)	结束整个程序的执行

使用 os 库模块的程序示例如图 3-14 所示。在该程序中，第 3 行引入了 os 库模块。第 5～7 行输出了两行“我喜欢 Python”和一个空白行。第 9 行调用 system 函数暂停程序执行。第 11 行调用 system 函数清除了之前输出的内容。第 13 行输出了一行“我喜欢 Python”。第 15 行再次调用 system 函数暂停程序执行。第 17 行调用_exit 函数退出了程序运行。很显然，第 19 行代码是无法执行的。

```
File  Edit  Format  Run  Options  Window  Help
 1 #使用os模块中的函数
 2
 3 from os import *    #引入os模块
 4
 5 print("我喜欢Python")
 6 print("我喜欢Python")
 7 print()
 8
 9 system("Pause")     #调用system函数暂停程序执行
10
11 system("cls")       #调用system函数清屏
12
13 print("我喜欢Python")
14
15 system("Pause")     #调用system函数暂停程序执行
16
17 _exit(1)            #调用_exit函数结束程序运行
18
19 print("我喜欢Python")
20
```

(a) 程序代码

```
C:\WINDOWS\py.exe
我喜欢Python
我喜欢Python

请按任意键继续. . .
```

```
C:\WINDOWS\py.exe
我喜欢Python
请按任意键继续. . .
```

(b) 运行效果

图 3-14 使用 os 库模块的程序示例

除了 1.4 节介绍的在 IDLE 环境中运行程序的方法外，还可以通过直接双击程序文件图标的方式来运行程序。使用双击文件图标方式运行程序时，程序的运行结果输出到了命令窗口中。通常情况下，程序运行结束时命令窗口会自动关闭，导致无法浏览程序的输出情况，这种现象叫作闪屏。

解决闪屏的常用办法有以下两种。

- 在程序末尾添加语句：system("pause")。
- 在程序末尾添加语句：input("按回车键继续…")。

有关闪屏及其处理方法，视频 3.2.2 中有详细的演示和介绍。

2. math 库

math 库提供了用于数学运算的常数及常用函数。

math 库中的常数如表 3-7 所示。

表 3-7　math 库中的常数

常　数	数学表示	描　述
math.pi	π	圆周率，值为 3.141592653589793
math.e	e	自然对数，值为 2.718281828459045
math.inf	∞	正无穷大，负无穷大为-math.inf
math.nan		非浮点数标记，nan(非数值)

math 库中的常用数值运算函数如表 3-8 所示。

表 3-8　math 库中的常用数值运算函数

常　数	数学表示	描　述
math.fabs(x)	$\|x\|$	返回 x 的绝对值
math.fmod(x,y)	$x\%y$	返回 x 与 y 的余数
math.fsum($x1$,$x2$,…)	$x1+x2+\cdots$	返回浮点数的和
math.ceil(x)	$\lceil x \rceil$	返回不小于 x 的最小整数
math.floor(x)	$\lceil x \rceil$	返回不大于 x 的最大整数
math.factorial(x)	$x!$	返回 x 的阶乘
math.gcd(x,y)		返回 x 与 y 的最大公约数
math.frexp(x)	$x=m*2^e$	返回(m,e)
math.ldexp(x,y)	$x*2^y$	返回 $x*2^y$
math.modf(x)		返回 x 的小数和整数部分
math.trunc(x)		返回 x 的整数部分
math.copysign(x,y)	$\|x\|*\|y\|/y$	用 y 的正负号替换 x 的正负号
math.isclose(x,y)		判断 x 和 y 的相似性
math.isfinitde(x)		判断 x 是有限还是无限
math.isinf(x)		判断 x 是否为无穷大
math.isnan(x)		判断 x 是否为非数值(nan)

math 库中常用的三角运算函数如表 3-9 所示。

表 3-9　math 库中常用的三角运算函数

常　数	数学表示	描　述
math.dgree(x)		把用弧度表示的 x 转换为角度值
math.ridans(x)		把用角度表示的 x 转换为弧度值
math.hypoot(x,y)		返回(x,y)到坐标原点的距离
math.sin(x)	$\sin x$	返回弧度 x 的正弦值
math.cos(x)	$\cos x$	返回弧度 x 的余弦值
math.tan(x)	$\tan x$	返回弧度 x 的正切值
math.asin(x)	$\arcsin x$	返回弧度 x 反正弦函数值
math.acos(x)	$\arccos x$	返回弧度 x 反余弦函数值
math.atan(x)	$\arctan x$	返回弧度 x 反正切函数值
math.atan2(y,,x)		返回 y/x 的反正切函数值
math.sinh(x)	$\sinh x$	返回 x 的双曲正弦函数值
math.cosh(x)	$\cosh x$	返回 x 的双曲余弦函数值
math.tanh(x)	$\tanh x$	返回 x 的双曲正切函数值
math.asinh(x)	$\operatorname{arcsinh} x$	返回 x 的反双曲正弦函数值
math.acosh(x)	$\operatorname{arccosh} x$	返回 x 的反双曲余弦函数值
math.astanh(x)	$\operatorname{arctanh} x$	返回 x 的反双曲正切函数值

math 库中的常用幂及对数运算函数如表 3-10 所示。

表 3-10　math 库中的常用幂及对数运算函数

常　数	数学表示	描　述
math.pow(x,y)	x^y	返回 x^y
math.exp(x)	e^x	返回 e^x
math.expm1(x)	e^x-1	返回 e^x-1
math.sqrt(x)	$\sqrt{x}$	返回$\sqrt{x}$
math.log(x[,base])	$\log_{base}x$	返回底数为 base 的 x 的对数值
math.log1p(x)	$\ln(1+x)$	返回 $1+x$ 的自然对数值
math.log2(x)	$\log x$	返回以 2 为底 x 的对数值
math.log10(x)	$\log_{10}x$	返回以 10 为底 x 的对数值

math 库中的 3 个特殊运算函数如表 3-11 所示。

表 3-11　math 库中的 3 个特殊运算函数

常　数	数 学 表 示	描　述
math.erf(x)	$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt$	高斯误差函数
math.erfc(x)	$\frac{2}{\sqrt{\pi}}\int_{0x}^{\infty} e^{-t^2}\,dt$	余补高斯误差函数
math.gamma(x)	$\int_{0x}^{\infty} x^{t-1}e^{-x}\,dt$	欧拉第二积分函数


扫一扫

3.2.5　3 个程序设计实例

1. 身份识别

【实例 3-1】　编程实现，提示用户输入姓名，选择性别，根据输入的姓名及选择的性别（“男”或“女”）输出不同的问候语。如果性别选择“男”，则输出“欢迎**先生!”，否则就输出“欢迎**女士!”。

经过分析确定了程序的数据结构，需要用到 4 个 string 类型的变量——msg、name、sex 和 greeting，分别用来存储提示信息、姓名、性别和问候语。

该程序算法流程图如图 3-15 所示。

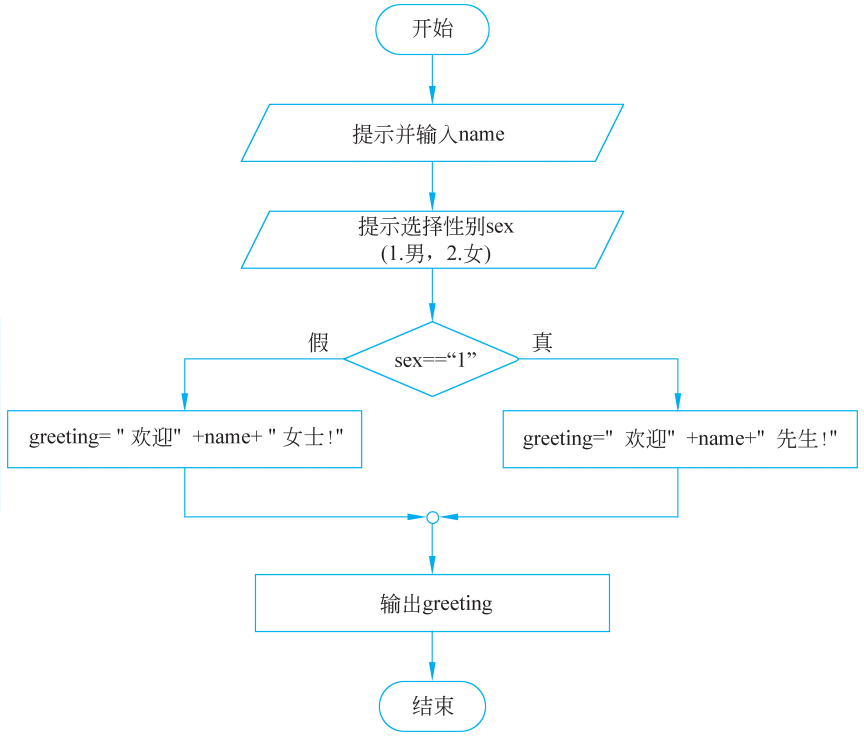


图 3-15　【实例 3-1】的算法流程图

程序的完整代码如图 3-16 所示。

```
File  Edit  Format  Run  Options  Window  Help
#根据输入和选择输出问候语的程序
msg = '''
    请选择性别(男或女)
    ---------------------------
    选择"男",请输入1
    选择"女",请输入2
    ---------------------------
    请输入你的选择(1或2):    '''

name = input("\n请输入姓名:  ")  #提示输入姓名
sex  = input(msg)                #提示选择性别

#双路分支结构
if sex =="1":
    greeting = "欢迎"+name+"先生!"
else:
    greeting = "欢迎"+name+"女士!"


#输出结果
print()             #输出空白行
print(greeting)
print()             #输出空白行
```

图 3-16 【实例 3-1】的程序代码

在该程序中,第 2～8 行定义了变量 msg,并给它赋值一个多行的字符串。第 10 行把提示输入的姓名存在变量 name 中。第 11 行使用 msg 作参数,把提示输入的性别存在变量 sex 中。第 14～17 行是 if-else 双路分支结构,实现根据输入的性别生成不同的问候语。

本程序的运行情况,视频 3.2.3 中有详细演示。

2. 求算术平方根

【实例 3-2】 编程实现,提示用户输入一个整数,输出这个整数和它的算数平方根。

经过对问题的分析,确定了程序的数据结构,需要用到两个变量——x、sqrX,分别用来存储输入的整数和与其对应的算术平方根。

该程序算法流程图如图 3-17 所示。

程序的完整代码和运行效果,如图 3-18 所示。

在该程序中,第 3 行是引入 math 模块的语句。第 6 行是调用 eval 和 input 函数给 x 输入数据的语句。第 9～13 行是 if-else 双路分支结构,用来实现根据 x 的值做不同的处理。第 11 行和第 13 行用到了 2.5 节介绍的使用 format 方法控制输出数据的知识。

3. 求算术平方根的改版

【实例 3-3】 使用单路分支 if 语句编程实现与【实例 3-2】同样的功能。

该程序的数据结构与【实例 3-2】完全相同。

程序的算法流程图如图 3-19 所示。

程序的完整代码和运行效果如图 3-20 所示。

在该程序中,第 4 行是引入 os 模块的语句。第 7 行是调用 eval 和 input 函数给 x 输入数据的语句。第 10～12 行是单路 if 分支结构,用来实现若 x 的值为负数,则控制结束程序。第 12 行通过调用 os 模块的_exit 函数结束程序运行。很显然,如果 if 后面的条件 $x<0$ 成立,则执行 if 后的第 11 行和第 12 行,第 15 行和第 16 行执行不了,反之,若 $x<0$ 的条件不成立,if 后的第 11 行和第 12 行执行不了,就会执行第 15 行和第 16 行。

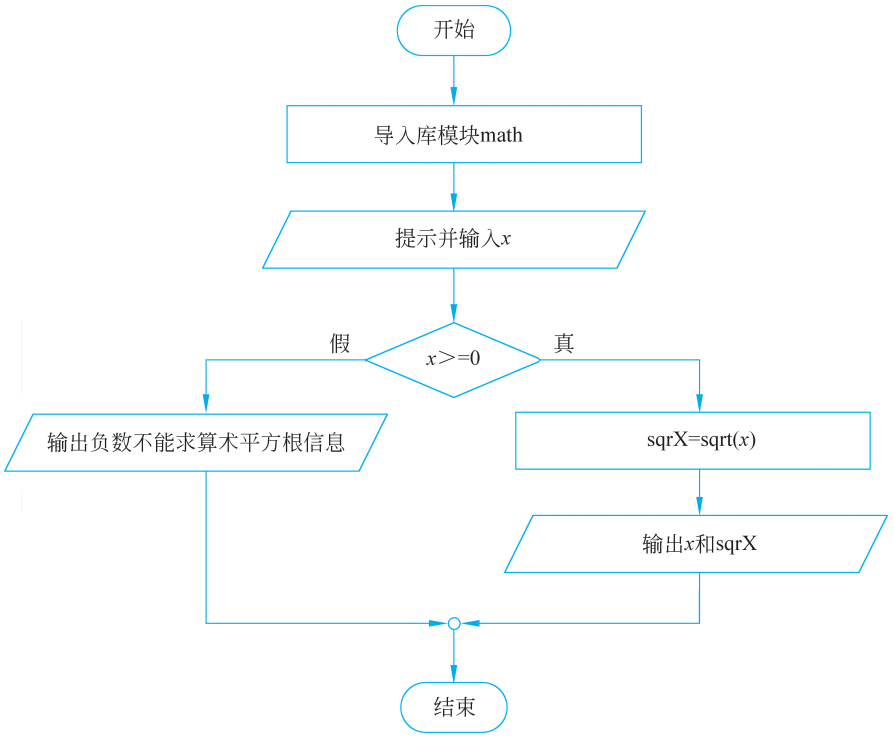


图 3-17 【实例 3-2】的算法流程图

```
File  Edit  Format  Run  Options  Window  Help
#输入x求算术平方根输出

from math import *   #引入库模块math


#提示输入x
x = eval(input("\n请输入一个整数:  "))

#双路分支结构
if x>=0:
    sqrX = sqrt(x)
    print("\n{}的算术平方根是{}\n".format(x,sqrX))
else:
    print("\n负数{}不能求算术平方根! \n".format(x))
```

```
请输入一个整数:  2
2的算术平方根是1.4142135623730951
>>>
```

```
请输入一个整数:  -1
负数-1不能求算术平方根!
>>>
```

(a) 程序代码　　(b) 两次运行效果

图 3-18 【实例 3-2】的程序代码及运行效果

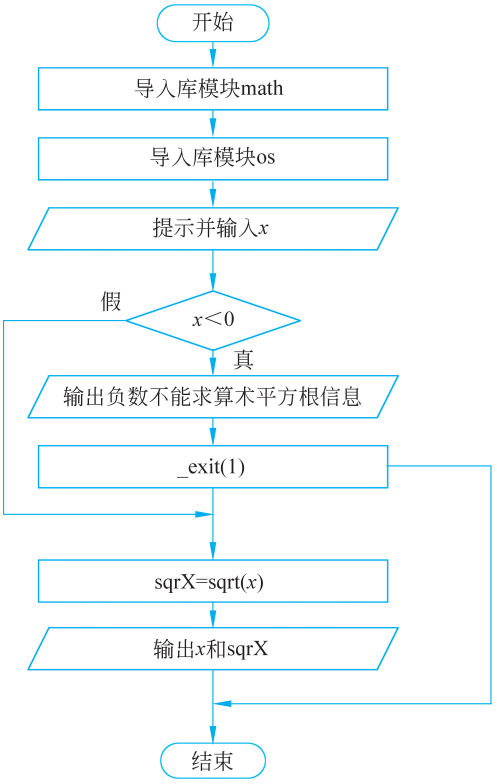


图 3-19 【实例 3-3】的算法流程图

```
File  Edit  Format  Run  Options  Window  Help
#输入x求算术平方根输出

from math import *    #引入库模块math
import os             #引入库模块os

#提示输入x
x = eval(input("\n请输入一个整数:  "))

#单路分支结构
if x<0:
    print("\n负数{}不能求算术平方根! \n".format(x))
    os._exit(1)       #结束整个程序

#求平方根及输出结果的语句
sqrX = sqrt(x)
print("\n{}的算术平方根是{}\n".format(x,sqrX))
```

(a) 程序代码

```
请输入一个整数:  2
2的算术平方根是1.4142135623730951
>>>
```

```
请输入一个整数:  -1
负数-1不能求算术平方根!
>>>
```

(b) 两次运行效果

图 3-20 【实例 3-3】的程序代码及运行效果

3.3 复杂分支程序设计

扫一扫

在实际编程时，很多问题的处理往往需要从多个选项中选择一个执行，这样的程序叫作复杂分支程序。通过分支语句的嵌套可以实现多路分支程序设计。

3.3.1 分支语句的嵌套

Python 允许 if-else 与 if-else 之间、if 与 if-else 之间以及 if 与 if 之间进行相互嵌套。这种分支语句中套着分支语句的结构叫作分支语句的嵌套。分支语句的嵌套构成了语句之间的包含和层次关系，处于最外面的是第 1 层，包含在第 1 层里的是第 2 层……依此类推。

if-else 语句的 3 层嵌套结构如图 3-21 所示。

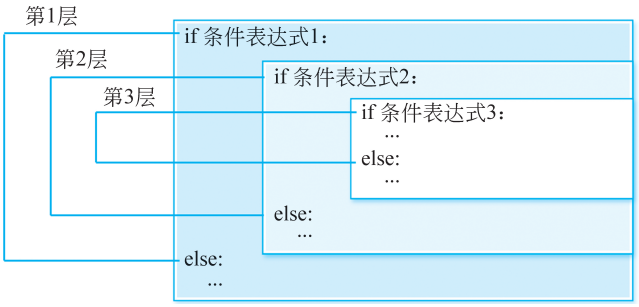
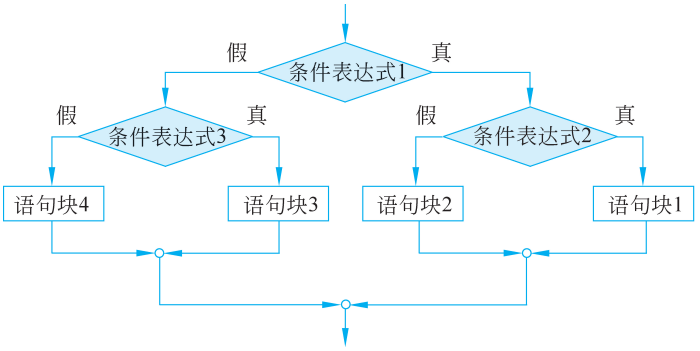


图 3-21 if-else 语句的 3 层嵌套结构

通过分支语句的嵌套可以实现多路分支程序设计。

使用 if-else 嵌套实现 4 路分支的流程图与语句结构，如图 3-22 所示。



语句结构

```
if 条件表达式1：
    if 条件表达式2：
        语句块1
else：
    语句块2
    if 条件表达式3：
        语句块3
else：
    语句块4
```

图 3-22 使用 if-else 嵌套实现 4 路分支的流程图与语句结构

上述结构的执行过程是：如果表达式 1 和表达式 2 同时成立，则执行语句块 1，结束整个结构；如果表达式 1 成立，表达式 2 不成立，则执行语句块 2，结束整个结构；如果表达式 1 不成立，表达式 3 成立，则执行语句块 3，结束整个结构；如果表达式 1 和表达式 3 都不成立，则执行语句块 4，结束整个结构。

使用 if-else 嵌套实现 3 路分支的程序示例，如图 3-23 所示。其中，图 3-23(b)是在

IDLE 环境下分别运行程序 3 次的效果截图。

```
File  Edit  Format  Run  Options  Window  Help
 1 #if-else嵌套结构
 2
 3 x=eval(input("请输入一个数:  "))
 4
 5 if x>0:
 6     print("输入的{}是正数".format(x))
 7 else:
 8     if x<0:
 9         print("输入的{}是负数".format(x))
10     else:
11         print("输入的是零")
12
13 input("按回车键结束...")
14
```

(a) 程序代码

```
请输入一个数:  10
输入的10是正数
按回车键结束...
```

```
请输入一个数:  -1
输入的-1是负数
按回车键结束...
```

```
请输入一个数:  0
输入的是零
按回车键结束...
```

(b) 3次运行结果

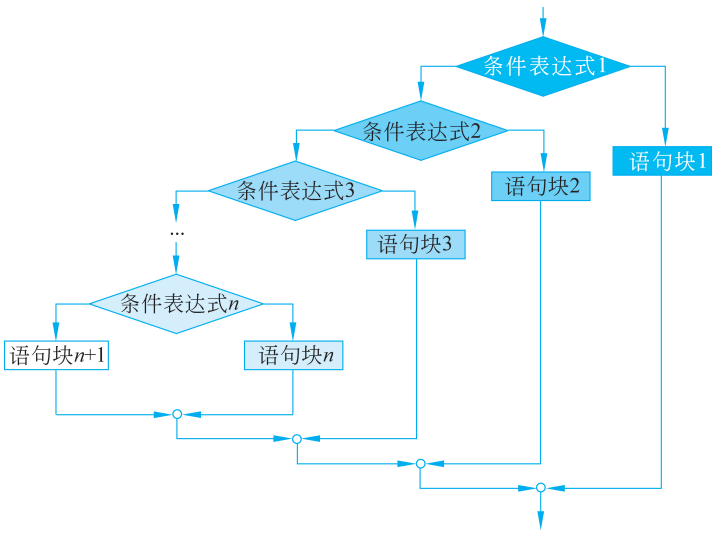
图 3-23　使用 if-else 嵌套实现 3 路分支的程序示例

在该程序中，第 5～11 行是 if-else 的两层嵌套结构，实现根据 x 的值从 3 个 print 语句中选择一个执行。第 13 行用到了 3.2.4 节介绍的使用 input 语句防止在双击文件图标方式运行程序时出现的闪屏问题。

3.3.2　多路分支语句 if-elif-else

从使用 if-else 语句嵌套实现多路分支的程序结构不难看出，由于程序要写成缩进形式，当嵌套的层次较多时，后面的语句就会向右缩进较远的距离。如果语句太长，就可能超出屏幕显示范围，给浏览代码造成困难。为了解决这一问题，Python 提供了 if-elif-else 语句，使用它可以非常方便地实现多路分支程序设计。

if-elif-else 语句的简单流程图和语句格式如图 3-24 所示。



语句结构

```
if 条件表达式1:
    语句块1
elif 条件表达式2:
    语句块2
elif 条件表达式3:
    语句块3
...
elif 条件表达式n:
    语句块n
else:
    语句块n+1
```

图 3-24　if-elif-else 语句的简单流程图和语句格式

该结构的执行过程是：首先处理 if 后面的条件表达式 1，如果结果为“真”(True)，则执行 if 后的语句块 1，结束整个结构；如果结果为“假”(False)，则处理 elif 后的条件表达式 2，

如果结果为“真”(True)则执行 elif 后的语句块 2,结束整个结构……这样一直进行下去。如果前面的 $n-1$ 个条件都不成立,则处理最后一个 elif 后的条件表达式 n,如果结果为“真”(True),则执行其后面的语句块 n,结束整个结构;如果前面的 n 个条件都不成立,则执行 else 后的语句块 $n+1$,结束整个结构。很显然,该结构是根据条件的“真”(True)或“假”(False),从 $n+1$ 中情况中选择了一个执行。

使用 if-elif-else 实现多路分支的程序示例如图 3-25 所示。

```
File  Edit  Format  Run  Options  Window  Help
 1 #if-elif-else语句
 2
 3 x=eval(input("请输入一个数:  "))
 4
 5 if x>0:
 6     print("输入的{}是正数".format(x))
 7 elif x<0:
 8     print("输入的{}是负数".format(x))
 9 else:
10     print("输入的是零")
11
12 input("按回车键结束...")
13
```

(a) 程序代码

```
请输入一个数:  10
输入的10是正数
按回车键结束...

请输入一个数:  -1
输入的-1是负数
按回车键结束...

请输入一个数:  0
输入的是零
按回车键结束...
```

(b) 三次运行效果

图 3-25　使用 if-elif-else 实现多路分支的程序示例

该程序与图 3-23 中的程序实现的功能完全一样。在该程序中,第 5～10 行是使用 if-elif-else 语句实现的 3 路分支结构,实现根据 x 的值从 3 个 print 语句中选择一个执行。由于 if、elif、else 都处在同一列上,不存在缩进问题,所以克服了 if-else 嵌套结构嵌套层次较多时向右缩进太多的不足。

3.3.3　程序异常处理

程序异常是指程序在运行过程中由于异常情况而引发的运行突然终止和报错的现象。图 3-26 所示的程序示例中,因为用户输入的数据不是整数而引发了程序运行出错的问题,这就是程序异常。

程序异常会破坏程序运行的稳定性,影响用户正常使用程序的体验,因此必须对程序中的异常进行处理。

```
请输入一个整数:  aa
Traceback (most recent call last):
  File "D:\py\3-7.py", line 7, in <module>
    x = eval(input("\n请输入一个整数:  "))
  File "<string>", line 1, in <module>
NameError: name 'aa' is not defined
>>>
```

图 3-26　发生异常的程序示例

在 Python 中,使用关键字 try 和 except 处理异常。其基本语法格式如下。

```
try:
    语句块 1
```

```
except:
    语句块 2
```

其中，语句块 1 是可能出现异常的程序代码，语句块 2 是异常发生后做出处理的代码。在程序执行时，若语句块 1 出现异常情况，就会跳转到语句块 2 去执行。

对【实例 3-2】进行异常处理的程序示例如图 3-27 所示。

```
File  Edit  Format  Run  Options  Window  Help
#输入x求算术平方根输出

from math import *    #引入库模块math

#异常处理
try:
    #提示输入x
    x = eval(input("\n请输入一个整数:  "))

    #双路分支结构
    if x>=0:
        sqrX = sqrt(x)
        print("\n{}的算术平方根是{}\n".format(x,sqrX))
    else:
        print("\n负数{}不能开算术平方根！\n".format(x))
except:
    print("\n输入了非数值型数据，请检查！")
```

```
请输入一个整数:  aa

输入了非数值型数据，请检查！
>>>
```

图 3-27　程序异常处理的程序示例

比较两个程序的运行结果可以看出，与之前程序不同的是，当输入非法数据时，程序运行不再突然终止和报错，而是跳转到 except 执行其后面的语句，输出一条有意义的提示信息，然后正常结束程序。


扫一扫

3.3.4　3 个程序设计实例

1. 字符识别

【实例 3-4】　使用分支语句的嵌套结构编程，实现把提示用户输入的一个文字或符号存到 ch 中，按下列要求输出：

① 若 ch 中是字母，则输出“输入的**是字母”。

② 若 ch 中是数字字符，则输出“输入的**是数字”。

③ 若 ch 中既不是字母也不是数字字符，则输出“输入的**是其他字符”。

经过对问题的分析，确定了它是一个三选一的分支结构，可以使用两层 if-else 嵌套实现。这个程序的数据结构很简单，只需要一个 string 型的变量 ch，用来存储用户输入的内容。

该程序算法流程图如图 3-28 所示。

程序的完整代码如图 3-29 所示。

在该程序中，第 7～13 行是 if-else 两层嵌套结构，实现三路分支。第 10～13 行是内层 if-else 结构。在程序执行时，若第 7 行中 if 后面的条件成立，则执行第 8 行，结束程序；若第 7 行 if 后面的条件不成立，则再判断第 10 行 if 后的条件，若条件成立，则执行第 11 行，结束程序；若条件不成立，就执行第 13 行，结束程序。

本程序的运行情况，视频 3.3.2 里有完整演示。