第3章

数据处理编程

本章概要

根据代码的功能，可以将代码划分成不同函数和库形式的代码块，即对代码进行模块化管理。本章围绕批量数据类型，主要介绍如何利用程序设计方法，将数据与计算统一在一起，讲解同样的问题和数据，可以采用不同的程序结构和算法实现；同时学习模块化编程中必须掌握的方法和技巧。

学习目标

通过本章的学习，要求达到以下目标：

（1）掌握并会运用分支结构解决问题。

（2）掌握并会运用循环结构解决问题。

（3）掌握函数的定义和调用。

（4）掌握文件的读写操作。

（5）掌握数据预处理和数据标准化操作。

3.1 程序的基本结构

3.1.1 控制流

算法的控制结构章节介绍了算法的三种基本结构：顺序结构、选择结构和循环结构。在程序中，根据表达式求值的结构或判断条件，程序可以决定跳过某些指令、重复某些指令或者从几条指令中选择一条执行。算法的程序实现，实际上是事务处理的流程在计算机中的表现形式，很少有程序从第一条指令开始，简单地顺序执行每一条指令直至最后。“控制流语句”决定程序的处理路径。

图 3-1 是判断是否可以进入图书馆的流程图。根据判断框的条件和箭头指示的路径，能够了解从开始到结束的所有步骤。判断框中的条件相当于程序中的“控制流语句”，它将决定下一步要走的路径。条件框中，条件的判断结果用“是”和“否”表示。在 Python 程序中实现“控制流语句”，需要先学习布尔值、比较运算符和逻辑运算符。

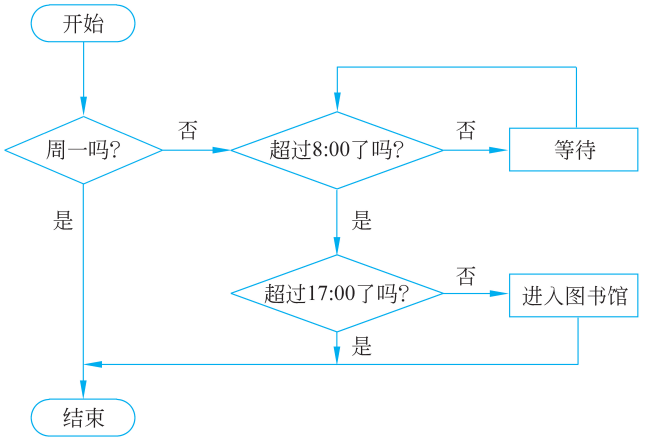

图 3-1　一张流程图，告诉你是否可以进入图书馆

1. 布尔值

布尔数据类型只有两种值：True 和 False。在 Python 中，必须以大写字母 T 和 F 开头，后面的字母是小写。布尔值可以用在表达式中，也可以保存在变量中，使用时注意不需要加引号。此外，True 或 False 不能作为变量名。图 3-2 显示了布尔值的一些正确和错误的使用方法。

```
>>> blnVar1 = True
>>> blnVar2 = False
>>> blnVar1
True
>>> blnVar2
False
>>> blnVar3 = true
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    blnVar3 = true
NameError: name 'true' is not defined
>>> blnVar4 = FALSE
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    blnVar4 = FALSE
NameError: name 'FALSE' is not defined
>>> True = 23*18
SyntaxError: cannot assign to True
```

图 3-2　布尔值的用法示例

2. 比较运算符

比较运算符又称为关系运算符，用来比较两个值，其结果是一个布尔值。用比较运算符可以构成简单的布尔表达式。表 3-1 列出了 Python 中的比较运算符。

表 3-1　比较运算符

比较运算符	说　明	比较运算符	说　明
==	相等	>	大于
!=	不相等	<=	小于或等于
<	小于	>=	大于或等于

通过对比较运算符两边的数值进行比较，可以得到一个布尔类型的值。

＝＝和！＝比较运算符用于任何数据类型的比较，判断运算符两边的数值是否相等。需要注意的是，整型或浮点型的值与字符串类型是不相等的。例如，在 Python 中 42 和'42'是不同的，所以关系运算 42＝＝'42'的结果是 False。图 3-3 是＝＝和！＝的运算示例。

```
>>> "Summer" == "Summer"
True
>>> 12300 == 12300
True
>>> 78.34 == 78.34
True
>>> 2.4 == 2.40
True
>>> True == True
True
>>> "Summer" == "summer"
False
>>> 42 == "42"
False
>>> "Summer" != "summer"
True
>>> 42 != '42'
True
>>> False != True
True
```

图 3-3　＝＝和！＝示例

在＝＝和！＝两边可以是其他表达式或变量，图 3-4 显示了数值和算术表达式的比较、变量和字符串的比较。

＞、＜、＞＝和＜＝运算符通常用于整型和浮点型数值的比较，也可以比较字符串、其他表达式或变量。图 3-5 为比较运算符的使用示例。

```
>>> 30 == 10+20
True
>>> strName = "xiaoming"
>>> strName == "xiaoming"
True
```

图 3-4　表达式或变量参与关系运算

```
>>> 20 < 21
True
>>> 100 > 99
True
>>> 20 <= 20
True
>>> 100 >= 100
True
>>> 'HelloKetty' < 'HelloWorld'
True
>>> 'a' > 'Z'
True
>>> x=10
>>> y=5
>>> x > 2 * y + 4
False
>>> 30 <= 5 * 7
True
```

图 3-5　＞、＜、＞＝和＜＝运算符示例

3. 逻辑运算符

Python 中有三个逻辑运算符：and、or 和 not。对布尔值进行比较，构成复合布尔表达式，形式如下。

布尔值/布尔表达式 逻辑运算符 布尔值/布尔表达式

在逻辑运算符两边可以直接是布尔值，也可以是简单的布尔表达式。

(1) and 运算符：二元运算符。and 运算符左右两边的布尔值/布尔表达式的值均为 True 时，整个复合布尔表达式的值为 True；and 两边的布尔值/布尔表达式有一方为 False，则复合表达式的结果就为 False。其运算规则如表 3-2 所示。

表 3-2 and 运算规则

布尔值/布尔表达式 1	布尔值/布尔表达式 2	and 结果
True	True	True
True	False	False
False	True	False
False	False	False

例如，图 3-6 的代码表示当 strWeekday 不是"Monday"、并且时间大于或等于 8 时，返回值为 True。

```
>>> strWeekday = "Tuesday"
>>> intTime = 10
>>> strWeekday != "Monday" and intTime >= 8
True
```

图 3-6 and 使用示例

可以用多个 and 构造更加复杂的复合布尔表达式。图 3-7 显示了满足进入图书馆的条件，即当 strWeekday 不是"Monday"，并且时间大于或等于 8 且小于或等于 17 时，返回值为 True，表示可以进入图书馆。

```
>>> strWeekday = "Tuesday"
>>> intTime = 10
>>> strWeekday != "Monday" and intTime >= 8 and intTime <= 17
True
```

图 3-7 多个 and 构成复杂的复合布尔表达式示例

(2) or 运算符：二元运算符。or 运算符左右两边的布尔值/布尔表达式的值均为 False 时，整个复合布尔表达式的值为 False；or 两边的布尔值/布尔表达式有一方为 True，则复合表达式的结果就为 True。其运算规则如表 3-3 所示。

表 3-3 or 运算规则

布尔值/布尔表达式 1	布尔值/布尔表达式 2	or 结果
True	True	True
True	False	True
False	True	True
False	False	False

例如，图 3-8 的代码表示当 strWeekday＝"Monday"、intTime＝7 时，各条件运算下的返回值。同样，可以使用多个 or 运算符构造更加复杂的复合布尔表达式。

（3）not 运算符：一元运算符。仅作用于一个布尔值/布尔表达式。其含义为取反，规则如表 3-4 所示。

表 3-4　not 运算规则

布尔值/布尔表达式	not 结果
True	False
False	True

图 3-9 为 not 运算的示例，同样，可以由多个 not 运算符构造复杂的复合布尔表达式。

```
>>> strWeekday = "Monday"
>>> intTime = 7
>>> strWeekday == "Monday" or intTime < 8
True
>>> strWeekday == "Monday" or intTime >= 8
True
>>> strWeekday != "Monday" or intTime < 8
True
>>> strWeekday != "Monday" or intTime >= 8
False
```

图 3-8　or 运算符示例

```
>>> intAge = 18
>>> not intAge < 18
True
>>> not True
False
>>> not not not not True
True
```

图 3-9　not 运算符示例

4. 多种运算符混合

在很多情况下，一个表达式中可能包含不同类型的运算符。Python 规定了不同类型运算符的优先顺序，表 3-5 描述了优先顺序从高到低的排序。

表 3-5　算术、关系和逻辑运算符的优先顺序

运算符类型	符号
算术运算符	**
	*，/
	+，—
关系运算符	<，<=，>，>=，==，!=
逻辑运算符	not
	and
	or

例 3-1：根据表 3-6 中变量 a，b，c 的值，填写复合布尔表达式的计算结果。

表 3-6　变量 a，b，c 的取值及复合布尔表达式

a	b	c	a>12 or c>b and c>=2	not(a>2 and b>a or c>1)
10	−6	2		
−9	3	−7		

分析：要计算复合布尔表达式的结果，在明确各运算符的优先顺序之后，可以借助图示分解法。

以第一行参数值为例，a=10，b=−6，c=2，其执行顺序和分步执行结果如图 3-10 所示。

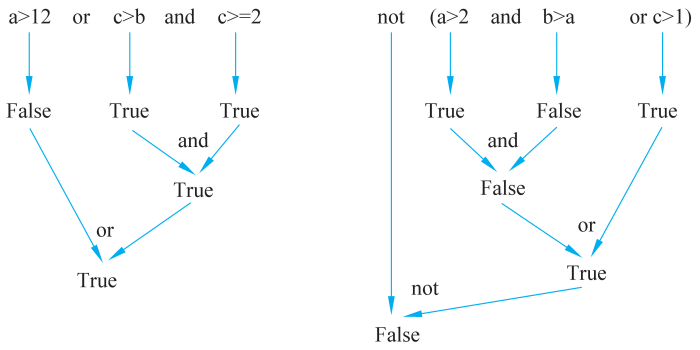

图 3-10　两个复合布尔表达式的执行过程

根据两个图中对运算执行顺序的分解，可以得到由第一行参数决定的两个复合布尔表达式的计算结果，分别为 True 和 False。同理，根据第二行参数计算得到的两个表达式结果为 False 和 True。因此，变量 a，b，c 的取值及复合布尔表达式结果如表 3-7 所示。

表 3-7　变量 a，b，c 的取值及复合布尔表达式结果

a	b	c	a>12 or c>b and c>=2	not(a>2 and b>a or c>1)
10	−6	2	True	False
−9	3	−7	False	True

3.1.2　程序运算选择

在程序实现中，最常见的控制流语句结构是选择结构，包括单分支选择、双分支选择和多分支选择。

1. if 语句——单分支选择结构

if 语句是最简单的选择结构，其执行流程图如图 3-11 所示。根据 if 语句的执行流程，如果代表条件的布尔表达式值为 True，则执行代码块的程序，否则跳过代码块。

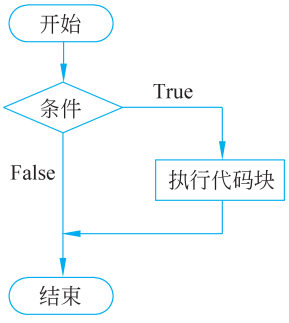

图 3-11　if 语句流程图

在 Python 中，if 语句的语法形式为：

```
if 布尔表达式：
    代码块
```

注意：Python 是第一批强制缩进的编程语言之一。Python 通过缩进表示哪几条语句是一个语句分组中的一部分。缩进组被称为“语句块”或“代码块”。在其他语言中，缩进是一种良好的编写方式，然而在 Python 中，缩进是强制性的，如果错误使用缩进或漏用缩进，都会造成程序语义上的错误。代码块语法有以下五个简单的规则需要遵守。

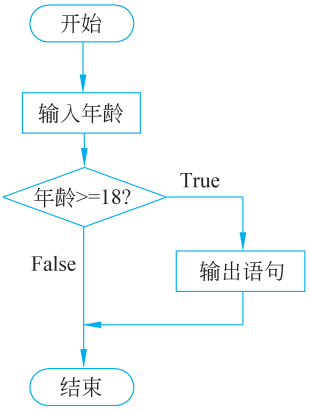

图 3-12　例 3-2 流程图

（1）代码块前一行的语句总是以冒号(:)字符结束。

（2）从代码块第一行开始，下面的所有代码行必须缩进。

（3）代码块可以包含其他代码块。

（4）缩进增加时，新的代码块开始。

（5）缩进减少为零，或与上一层的代码块对齐，则代码块结束。

例 3-2：编写程序，实现功能如下。

（1）接收输入的年龄。

（2）当年龄大于或等于 18 时，输出“Congratulations! You can apply for a driver's license.”。

分析：年龄是否满足 18 岁是判断条件，当大于或等于 18 岁时，执行输出的操作，处理流程图如图 3-12 所示。

根据流程图，Python 程序实现如下。

```
intAge = int(input("Enter your age:"))
if intAge >= 18:
        print("Congratulations! You can apply for a driver's license.")
```

该程序的执行结果如图 3-13 所示：当输入的年龄为 18 岁时，输出字符串。

```
Enter your age:18
Congratulations! You can apply for a driver's license.
```

图 3-13　条件为真时，输出字符串

当输入年龄为 16 岁时，不会输出字符串，如图 3-14 所示。

```
Enter your age:16
>>>
```

图 3-14　条件为假时，不输出字符串

2. if-else 语句——双分支选择结构

在 Python 中，if-else 语句的语法形式为：

```
if 布尔表达式:
    代码块 1
else:
    代码块 2
```

与 if 语句相比，if-else 语句结构有两条路径(True 和 False)，每条路径上都可以执行代码块，其执行流程图如图 3-15 所示。根据 if-else 语句的执行流程，如果代表条件的布尔表达式值为 True，则执行代码块 1 的程序，否则执行代码块 2 的程序。

例 3-3：编写程序，实现功能如下。

（1）接收输入的整数。

（2）判断该整数能够被 2 整除，则输出“偶数”；否则，输出“奇数”。

分析：输入的整数是否能被 2 整除是判断条件，处理流程图如图 3-16 所示。

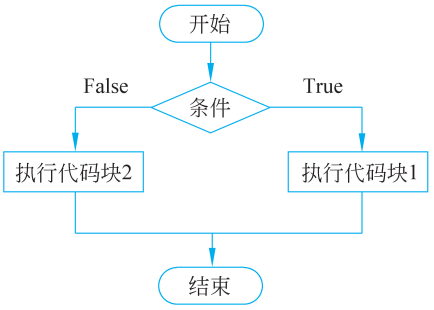

图 3-15　if-else 语句流程图

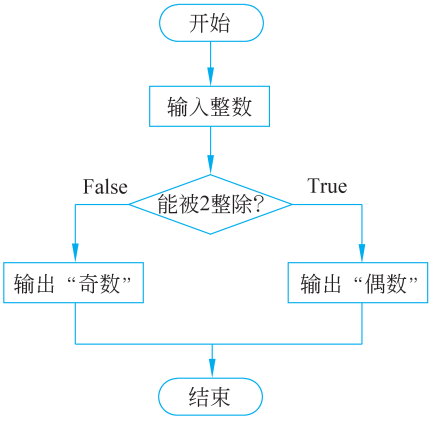

图 3-16　例 3-3 流程图

根据流程图，Python 程序实现如下。

```
intNum = int(input("请输入一个整数:"))
if intNum % 2 == 0:
    print(intNum,"是一个偶数")
else:
    print(intNum,"是一个奇数")
```

该程序的执行结果如图 3-17 所示：分别输入 37 和 6802 时，执行不同的代码块，输出不同的信息。

根据条件判断的结果，每个分支上可以执行一个代码块。

```
==============
请输入一个整数:37
37 是一个奇数
>>>
================ RESTART:
==============
请输入一个整数:6802
6802 是一个偶数
>>>
```

图 3-17　分别输入奇数和偶数时的执行结果

例 3-4：编写程序，根据输入的转换标志，实现摄氏度与华氏度的转换。

（1）输入温度转换标志位，该标志只能为“0”或“1”：“0”表示将摄氏度转换为华氏度；“1”表示将华氏度转换为摄氏度。如果输入的标志位不是“0”或“1”，则提示错误信息，处理结束。

（2）输入温度值。

（3）如果输入的温度转换标志位为“0”，则将（2）输入的温度值转换为华氏度；否则，将（2）输入的温度值转换为摄氏度。

注：摄氏温度(C)和华氏温度(F)之间的换算方法为：F=9/5C+32，或 C=5/9(F−32)。

分析：该功能包括两个判断条件。首先，判断输入的温度转换标志位是否为“0”或“1”，这个判断可以通过 if 语句实现；其次，判断输入的温度转换标志位是否为“0”，若是“0”，则执行摄氏度转华氏度的处理；否则，执行华氏度转摄氏度的处理。

根据分析，Python 程序实现如下。

```
strFlg = input("请输入转换标志位：")
if strFlg != "0" and strFlg != "1":
    print("输入的标志位不正确！")
else:
    intTemp = int(input("请输入温度："))
    if strFlg == "0":
        fltF = round(9 / 5 * intTemp + 32,2)
        print("摄氏度",intTemp,"转换为华氏度为：",fltF)
    else:
        fltC = round(5 / 9 * (intTemp - 32),2)
        print("华氏度",intTemp,"转换为摄氏度为：",fltC)
```

该程序的执行结果如下：当输入“0”或“1”以外的标志位时，提示输入错误，如图 3-18 所示。

```
===============
请输入转换标志位：3
输入的标志位不正确！
>>>
```

图 3-18　输入“0”或“1”以外的标志位时提示输入错误

当输入标志位为“0”，再输入温度，进行摄氏度转换为华氏度的处理，最后输出转换后的华氏度。执行示例如图 3-19 所示。

```
===============
请输入转换标志位：0
请输入温度：42
摄氏度 42 转换为华氏度为： 107.6
```

图 3-19　输入标志位为“0”，执行摄氏度到华氏度转换的代码块

当输入标志位为“1”，再输入温度，进行华氏度转换为摄氏度的处理，最后输出转换后的摄氏度。执行示例如图 3-20 所示。

```
===============
请输入转换标志位：1
请输入温度：135
华氏度 135 转换为摄氏度为： 57.22
```

图 3-20　输入标志位为“1”，执行华氏度到摄氏度转换的代码块

3. if-elif 语句——多分支选择结构

在 Python 中，if-elif 语句的语法形式为：

```
if 布尔表达式 1:
    代码块 1
elif 布尔表达式 2:
    代码块 2
elif 布尔表达式 3:
    代码块 3
```

```
...
elif 布尔表达式 n:
    代码块 n
else:
    代码块 n+1
```

if-else 满足了有两条路径时的处理需求，但有时候程序需要处理有更多可能性的情况，每一种可能的情况，都有相对应的代码块需要执行，elif 即满足这种情况。if-elif 语句结构有多条路径，每一种可能性由 elif 后面的条件语句来判断，当满足该条件时，这条路径上的代码块就能够被执行。其执行流程图如图 3-21 所示，根据 if-elif 语句的执行流程，如果条件 1 满足，则执行代码块 1；否则，如果条件 2 满足，则执行代码块 2；……；否则，如果条件 n 满足，则执行代码块 n。条件 1～n 都不满足时，执行代码块 n＋1。

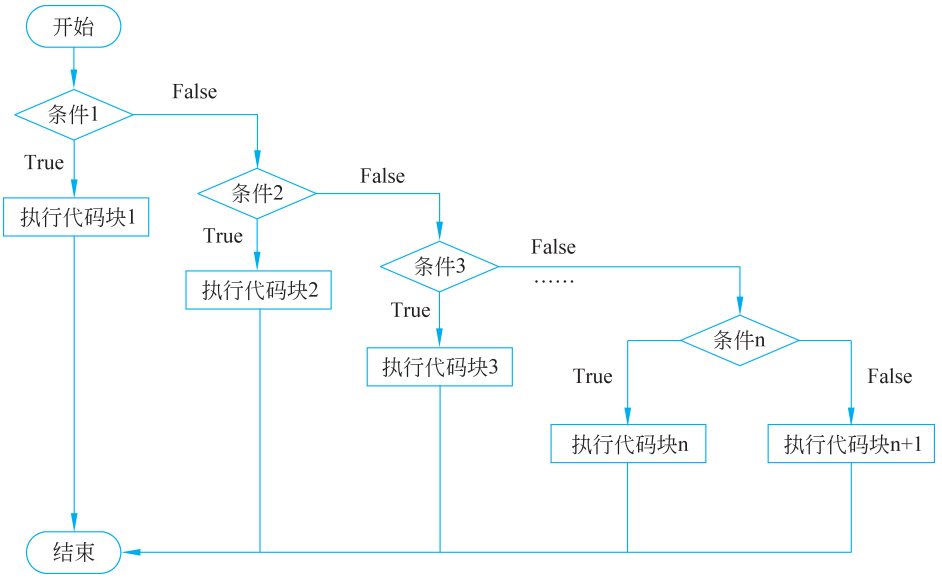

图 3-21　if-elif 语句流程图

例 3-5：编写程序，输入成年人的性别、身高和体重，计算 BMI 值，根据表 3-8 的 BMI 指标输出纤体程度。

（1）输入性别，性别只能为“男”或“女”。如果输入的性别不是“男”或“女”，则提示错误信息，处理结束。

（2）输入身高和体重。

（3）计算 BMI 值，计算公式为：$\mathrm{BMI}=\dfrac{\text{体重(kg)}}{\text{身高}(\mathrm{m}^2)}$

（4）根据表 3-8 输出相应的纤体程度。

分析：该功能包括两个判断条件。首先，判断输入的性别是否为“男”或“女”，这个判断可以通过 if 语句实现；其次，根据计算的 BMI 值输出相对应的纤体程度，由于 BMI 标准有多个程度，因此使用 if-elif 语句实现多分支判断。根据分析，Python 程序实现如下。

表 3-8　BMI 纤体标准

性别	程度	BMI 指标
女	过轻	＜19
	适中	[19,24)
	过重	[24,29)
	肥胖	[29,34)
	非常肥胖	≥34
男	过轻	＜20
	适中	[20,25)
	过重	[25,30)
	肥胖	[30,35)
	非常肥胖	≥35

```
strGender = input("请输入性别:")
if strGender != "男" and strGender != "女":
    print("输入的性别有误!")
else:
    fltWeight = float(input("请输入体重(单位为 kg):"))
    fltHeight = float(input("请输入身高(单位为 m):"))
    fltBMI = round(fltWeight / pow(fltHeight,2),2)
    print("纤体程度:",fltBMI)
    if strGender == "女":
        if fltBMI < 19:
            print("你有点弱不禁风哦!")
        elif fltBMI >= 19 and fltBMI < 24:
            print("多一两胖,少一两瘦,正正好!")
        elif fltBMI >= 24 and fltBMI < 29:
            print("对身材放松要求了吧!")
        elif fltBMI >= 29 and fltBMI < 34:
            print("最近吃得太任性了! 快打住!")
        elif fltBMI >= 34:
            print("以后一周只能吃一顿饭了!")
    else:
        if fltBMI < 20:
            print("你有点弱不禁风哦!")
        elif fltBMI >= 20 and fltBMI < 25:
            print("多一两胖,少一两瘦,正正好!")
        elif fltBMI >= 25 and fltBMI < 30:
            print("对身材放松要求了吧!")
        elif fltBMI >= 30 and fltBMI < 35:
            print("最近吃得太任性了! 快打住!")
        elif fltBMI >= 35:
            print("以后一周只能吃一顿饭了!")
```

该程序的执行结果如下：当输入“男”或“女”以外的标志位时，提示输入错误，如图 3-22 所示。

输入不同的性别、身高和体重，程序执行的结果示例如图 3-23 所示。

```
==========
请输入性别:无
输入的性别有误!
>>>
```

图 3-22　输入“男”或“女”以外的性别时提示输入错误

```
==========
请输入性别:男
请输入体重(单位为kg):56
请输入身高(单位为m):1.78
纤体程度: 17.67
你有点弱不禁风哦!
>>>
========== RESTART:
==========
请输入性别:男
请输入体重(单位为kg):78
请输入身高(单位为m):1.7
纤体程度: 26.99
对身材放松要求了吧!
>>>
========== RESTART:
==========
请输入性别:女
请输入体重(单位为kg):85
请输入身高(单位为m):1.6
纤体程度: 33.2
最近吃得太任性了! 快打住!
>>>
========== RESTART:
==========
请输入性别:女
请输入体重(单位为kg):130
请输入身高(单位为m):1.65
纤体程度: 47.75
以后一周只能吃一顿饭了!
>>>
```

图 3-23　根据性别、身高、体重不同代码块的执行结果示例

3.1.3　程序运算控制

本节介绍 Python 的循环控制结构。当需要代码块重复执行多次时，就要用到循环控制结构。Python 中使用 while 和 for 语句实现代码块的循环执行，同时可以用 break 和 continue 控制语句跳出循环。

1. while 循环语句

利用 while 语句，可以让一个代码块重复执行，其流程图如图 3-24 所示，while 语句表示当条件为 True 时，代码块就能够执行。

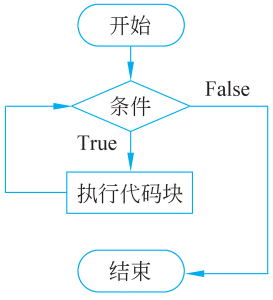

图 3-24　while 语句流程图

在 Python 中，while 语句的语法形式为：

```
while 布尔表达式：
    代码块
```

因为布尔表达式在进入循环之前会被计算，所以一个 while 语句可以执行多次循环，也可能执行零次循环。

例 3-6：编写程序，实现“重要的事情说 3 遍！”。

分析：可以直接输出三次“重要的事情说 3 遍！”实现该程序的功能，Python 代码如下。

```
print("重要的事情说 3 遍！")
print("重要的事情说 3 遍！")
print("重要的事情说 3 遍！")
```

这段程序的执行结果如图 3-25 所示。

在上面的代码中，三行代码完全相同，可以使用 while 循环，将一条 print 语句重复执行三次。Python 代码实现如下。

```
intCnt = 1
while intCnt <= 3:
    print("重要的事情说 3 遍！")
    intCnt = intCnt+1
```

这段程序的执行结果如图 3-26 所示，与原结果一样。在上述代码中，使用了 intCnt 变量来控制循环的次数：该变量初始值为 1，在进入 while 循环的时候，满足 intCnt＜＝3 的条件；此后每一轮循环，都会将该变量累加 1，每次进入循环，都会判断其值是否小于或等于 3；直到执行三轮循环之后，intCnt 累加到 4，不再满足循环条件，循环结束。

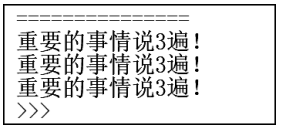

图 3-25　直接输出三次的执行结果

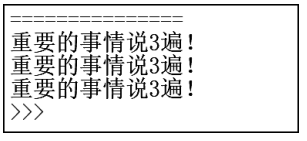

图 3-26　用 while 循环实现同一语句输出三次

使用 while 程序可以节省很多代码。例如，实现“重要的事情说 100 遍”的功能，用循环来实现将更加简单、灵活。

例 3-7：编写程序，用 while 循环语言实现数字 1～100 的累加求和。

分析：参考例 3-6，可以采用循环实现。设定一个变量控制循环的次数，变量初始值是 1，循环执行的条件是该变量值小于或等于 100。Python 代码如下。

```
intCnt = 1
intTotal = 0
while intCnt <= 100:
    intTotal = intTotal + intCnt
    intCnt = intCnt + 1
print("1~100 累加结果是:",intTotal)
```

在这段代码中，巧妙地利用 intCnt 变量实现了 1～100 的递增，使用变量 intTotal 进行累

加，最后输出累加结果如图 3-27 所示。

```
==============
1~100累加结果是：5050
>>> |
```

图 3-27　用 while 循环实现 1～100 的累加

2. for 循环语句

根据 while 循环语句的执行逻辑，当条件为 True 时，while 循环就会执行，因此 while 循环可以看作是不固定次数的循环。如果预先知道代码执行的固定次数，可以使用更加方便的 for 循环结构。

Python 中 for 语句的语法形式为：

```
for var in 序列:
    取下一项
    代码块
```

其中，var 是变量，它被依次赋予序列中的值，并且代码块针对每个值会执行一遍。for 语句执行流程如图 3-28 所示。

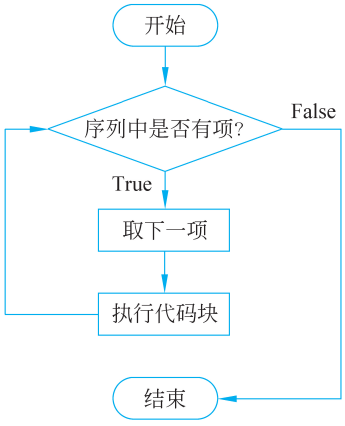

图 3-28　for 语句流程图

for 语句中的序列可以有多种形式。

1）列表形式

例如，用 for 循环实现输出 1、2、3、4、5 这五个数字，可以用列表的形式实现，代码如下。

```
for i in [1,2,3,4,5]:
    print(i)
```

在这段代码中，变量 i 将分别被赋予列表[1,2,3,4,5]中的值，并将其输出，因此 print(i)会执行 5 次。执行结果如图 3-29 所示。

2）字符串形式

可以用 for 循环实现输出一个字符串的各个字符，代码如下。

```
for strL in "Hello":
    print(strL)
```

在这段代码中，变量 strL 将分别被赋予字符串"Hello"的各个字符的值，并将其输出，因此 print(i)会执行 5 次。执行结果如图 3-30 所示。

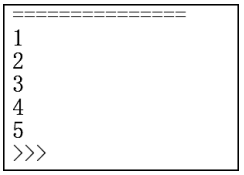

图 3-29　变量被赋予列表各项的值，for 语句执行结果

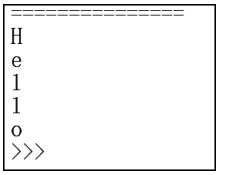

图 3-30　变量被赋予字符串各个字符的值，for 语句执行结果

3）range()函数形式

Python 的 range()函数可以用来创建一个整数序列，range()函数可以和 for 语句一起使用，实现固定次数的循环，其语法形式如下。

```
for var in range([initial_value,] final_value[,step]):
    代码块
```

- initial_value 是序列的起始值，这个参数是可选的，如果省略，则默认值为 0。
- final_value 是序列的终止值，但是不包括 final_value。
- step 是步长，表示每次循环后循环变量增加的值，这个参数也是可选的，如果省略，则默认值为 1。

上述输出 1、2、3、4、5 的程序，用 range()和 for 循环实现同样的功能，代码如下。

```
for i in range(5):
    print(i+1)
```

这段代码的执行结果与图 3-29 相同，都是输出了 1、2、3、4、5 这 5 个数字。这段代码中 range()函数的参数只有 5，是因为省略了 initial 和 step，相当于 range(0,5,1) 的形式。因为 initial_value 省略，所以初始值为 0，因此每次输出的值应该是序列的当前值＋1。如果初始值从 1 开始，可以这样实现：

```
for i in range(1,6):
    print(i)
```

在例 3-7 中，用 while 循环语言实现数字 1～100 的累加求和，现在使用 for 循环和 range()函数来实现同样的功能。

例 3-8：编写程序，用 for 循环语句和 range()函数实现数字 1～100 的累加求和。

分析：数字 1～100 的累加，即需要循环 100 次变量的累加求和，因此循环次数是固定的，可以使用 for 循环实现，代码如下。

```
intTotal = 0
for i in range(1,101):
    intTotal = intTotal + i
print("数字 1 到 100 的累加和是: ",intTotal)
```

在这段代码中，range(1,101) 表示从 1～100 执行 100 次循环，变量 i 的值也分别是 1～100，每次循环都对变量 i 进行累加，最后的执行结果如图 3-31 所示。

range()函数的 step 参数可以规定每次循环后循环变量增加的值，默认情况下，循环变

量每次增加 1。通过设定不同的 step，可以实现更加灵活的循环功能。

```
==============
数字1~100的累加和是： 5050
>>> |
```

图 3-31　用 for 循环实现 1～100 的累加

例 3-9：编写程序，用 for 循环语句和 range()函数输出 1～20 中的偶数。

分析：首先需要计算出循环的范围，查找范围是 1～20，由于 range()函数不包含终止值，所以 range()函数的参数是 1 和 21。判断偶数可以设定每次循环的步长为 2，从而跳过奇数，实现每隔一个数字(奇数)输出一个偶数。代码如下。

```
print("1 到 20 的偶数如下：")
for i in range(1,21,2):
    print(i+1)
```

在这段代码中，range(1,21,2) 表示从 1～20 以步长为 2 执行 10 次循环，由于输出的是偶数，所以每次需要输出变量 i+1，执行结果如图 3-32 所示。

有时候还可以使用负数作为步长，让循环计数逐渐减少，而不是增加。

例 3-10：编写程序，用 for 循环语句和 range()函数，逆序排列输出 20 到 1 中的奇数。

分析：循环次数和例 3-9 一样，从 20 到 1 的奇数有 10 个，且按倒序输出，因此可以设置序列的起始值为 20，终止值为 0，步长为－2，代码如下。

```
print("20 到 1 的奇数按倒序排列如下：")
for i in range(20,0,-2):
    print(i-1)
```

在这段代码中，range(20,0,2) 表示从 20 到 1 以步长为－2 执行 10 次循环，即每次变量 i 递减 2，由于输出的是奇数，所以每次需要输出变量 i－1，执行结果如图 3-33 所示。

```
==============
1~20的偶数如下：
2
4
6
8
10
12
14
16
18
20
>>> |
```

图 3-32　用 for 循环输出 1～20 中的偶数

```
==============
20~1的奇数按倒序排列如下：
19
17
15
13
11
9
7
5
3
1
>>>
```

图 3-33　用 for 循环输出 20～1 的奇数

3. break 语句

循环会消耗很多 CPU 资源，从程序的执行效率来讲，要谨慎使用循环。如果在某次循环中，根据指定条件需要及时跳出或者结束循环，而不是等到完成所有循环，可以使用循环终止语句。break 语句用在 while 和 for 循环中，用于中止循环，即循环条件没有达到 False 条件或者序列还没被完全递归完，也会停止执行循环语句，两者的流程图如图 3-34 所示。

例 3-11：编写程序，实现下列功能。

(1) 输入任意要查找的字符。

(2) 判定"Summer is here."字符串中是否含有上述(1) 输入的字符。

分析：通常的方法是遍历字符串，判断每一位字符是否等于要查找的字符。但有些情

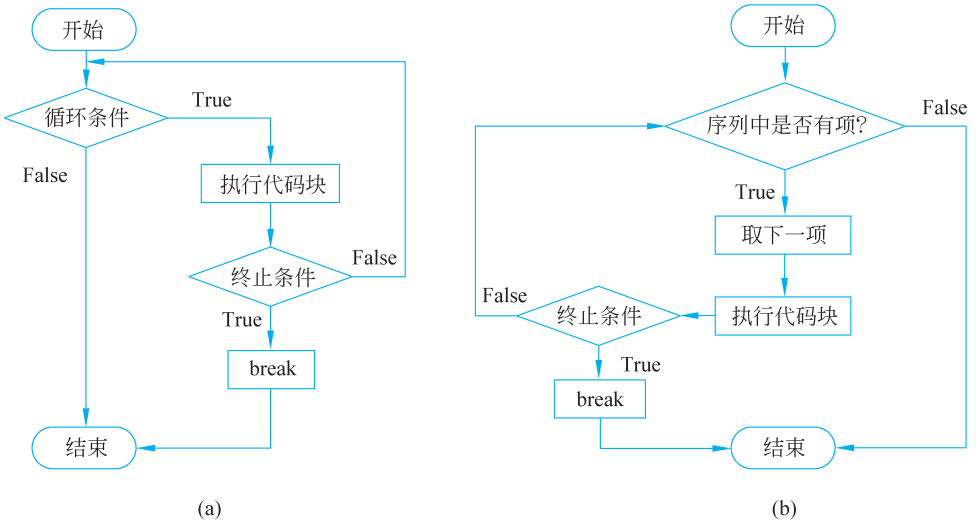

图 3-34　while 循环和 for 循环中有 break 的流程图

况下，无须执行完所有循环，就可以满足题目的要求。例如，输入的字符是"e"，在第五次循环的时候，就可以判断出包含字符"e"，而无须循环到最后，因此可以在 for 循环中加入一个终止条件，即一旦当前字符等于"e"，则可以判断出该字符串包含"e"，终止循环。代码实现如下。

```
strText = "Summer is here."
strFind = input("请输入要查找的字符: ")
blnFind = False
for strC in strText:
    if strC == strFind:
        blnFind = True
        break
if blnFind == True:
    print("找到了要查找的字符: ",strFind)
else:
    print("找不到要查找的字符: ",strFind)
```

在这段代码的 for 循环中，如果当前字符和待查找的字符一致，则利用 break 语句中止循环，执行 for 循环之后的语句。"Summer is here."字符串长度为 15，全部循环次数为 15，以查找字符是"e"为例，使用 break 语句能够节省 10 次循环的时间。上述代码的执行结果如图 3-35 所示。

```
==============
请输入要查找的字符: u
找到了要查找的字符:  u
>>>
================ RESTART:
==============
请输入要查找的字符: f
找不到要查找的字符:  f
>>>
```

图 3-35　查找到和未查找到的执行结果

在大规模数据处理中，提高程序的执行效率非常重要，我们要对每一个可能影响程序执行时间的代码进行优化，才能够应对大数据时代的数据处理需求。

4. continue 语句

和 break 语句类似，continue 语句在循环内部用于停止本轮循环。但 continue 语句是跳转至循环开始处，重新对循环条件进行判断或对循环序列求下一个值。

例 3-12：编写程序，用 for 循环、range()函数和 continue 语句输出 1～20 中的偶数。

分析：在例 3-9 中，使用 step＝2 实现了输出 1～20 中的偶数功能。如果不借助步长，而使用 continue 语句是否也能实现相同的功能？代码实现如下。

```
for i in range(1,21):
    if i%2 != 0:
        continue
    print(i)
```

代码的每次循环中都会判断当前值是否能被 2 整除，若不能被 2 整除，则执行 continue，即不执行 print(i)语句，结束本轮循环，直接跳转至循环开始处，取序列中的下一个值，进入下一次循环。程序流程图如图 3-36 所示。

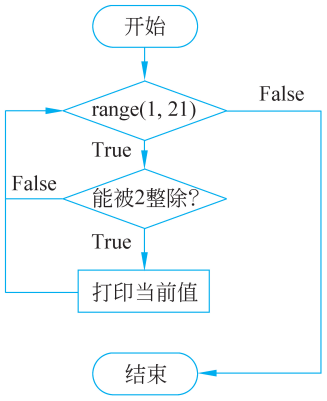

图 3-36　通过 continue 语句终止本轮循环

3.1.4　习题与实践

1. 填空题

（1）布尔数据类型只有两种值：_____和_____。

（2）比较运算符又称为关系运算符，用来比较两个值，其结果是一个_____。

（3）Python 中有三个逻辑运算符：_____、_____和_____。

（4）对于一个循环结构来说，终止整个循环的语句是_____，中止本轮循环的语句是_____。

2. 选择题

（1）下列不属于比较运算符的是（　　）。

A. ＝＝　　B. !＝　　C. ＞＝　　D. ＝

（2）下列布尔表达式的值为 True 的是（　　）。

A. 55＝＝"55" and 55＝＝55.0

B. 30＞＝90 or 55 ＝＝ "55"

C. 20 ＞ 4 * 4 and "a" ＞ "A" or 0＝＝0

D. "ABC" ＞ "abc"

（3）关于选择结构以下说法错误的是（　　）。

A. if 可以实现单分支选择结构

B. if-else 可以实现双分支选择结构

C. if-elif 可以实现多分支选择结构

D. if-else 和 if-elif 不可以嵌套使用

(4) 关于循环结构以下说法正确的是(　　)。

A. while 循环语句至少会执行一次循环

B. for 语句必须和 range()函数一起使用

C. range()的 step 参数可以是负数

D. range()的初始值参数一定要小于终止值参数

3. 判断题

(1) 当循环次数未知时,可以使用 for 循环结构。　(　　)

(2) 当循环结构已知时,不能使用 while 循环结构。　(　　)

(3) 在以下代码片段中,"Hello"显示 10 次。　(　　)

```
for i in range(1,10):
    print("Hello")
```

(4) 在 for 循环结构中,循环变量在每次循环的开始处被自动赋予序列中的后续值。

(　　)

(5) while 结构可能会执行 0 次循环。　(　　)

(6) 在以下代码片段中,"Hello"显示 10 次。　(　　)

```
i = 1
while i <= 10:
    print("Hello")
i = i + 1
```

3.2　模块化程序设计

在 Python 中,将用于实现某个功能的语句组合成一个模块,称为函数。本节将介绍函数的定义和使用方法。

3.2.1　函数

创建函数的目的是重复使用,提高程序开发效率。可以把函数看成是一个具有某种功能的“工具”,可以反复使用。

1. 创建函数

创建函数也称为定义函数。在 Python 中函数由关键字 def 来定义。函数定义的一般形式为:

```
def  函数名(参数列表):
函数体
return
```

其中，参数是可选的，用于指定向函数中传递的参数。如果有多个参数，参数之间用逗号分隔。如果无参数设置，则表明当前函数没有参数，调用时也无须指定参数。

根据需要，函数可以有返回值，用 return 返回。当函数无返回值时，可以省略 return 语句。

例 3-13：创建一个名为 Hello 的函数，其作用为输出“欢迎进入 Python 世界”的字符内容。

创建该函数的程序段如下。

```
def  Hello():
print("欢迎进入 Python 世界 ")
```

在程序中调用 Hello()函数，将显示“欢迎进入 Python 世界”的字符内容。

例 3-14：创建一个名为 sum()的函数，其作用是计算 n 以内的整数之和(包含 n)。下面为定义的函数程序段。

```
def sum(n):
    s=0
    for i in range(1,n+1):
        s=s+i
    return s
```

2. 调用函数

在 Python 中，在需要函数的地方，直接使用函数名来调用函数。如果定义的函数包含参数，则调用函数时也必须提供参数。

例 3-15：调用 sum(n)函数计算 1～50 的整数之和。

```
def sum(n):              #定义 sum 函数
    s=0
    for i in range(1,n+1):
        s=s+i
    return s
print(sum(50))           #调用 sum 函数,参数为 50
```

3. 形参与实参

在调用函数时，大多数情况下，主调函数和被调用函数之间有数据传递关系，这就是有参数的提取形式。函数参数的作用是传递数据给函数使用，函数利用接收的数据进行具体的操作处理。

函数参数在定义函数时放在函数名称后面的一对小括号中，在使用函数时，经常会用到形式参数(形参)和实际参数(实参)，两者都称为参数。

形式参数和实际参数在作用上的区别如下。

(1) 形式参数：在定义函数时，函数名后面括号中的参数为“形式参数”，也称形参。

(2) 实际参数：在调用一个函数时，函数名后面括号中的参数为“实际参数”。也就是将函数的调用方传递给函数的参数称为实际参数，也称实参。

根据实参的类型不同，可以分为传值和传地址两大类——将实参的值传递给形参和将实参的引用传递给形参两种情况。其中，当实参为不可变对象时，进行的是值传递；当形参