CHAPTER 第3章
判断与决策——选择程序设计

学习目标：

- 理解算法的概念和算法的描述方法。
- 理解关系运算、逻辑运算和条件运算。
- 掌握几种形式的逻辑判断条件。
- 能用选择结构解决逻辑判断与决策问题。
- 理解复合语句(语句块)的概念。
- 熟悉现有各种运算的优先级。

通过第2章的入门学习，大家已经能够用计算机解决一些比较简单的问题了。请大家先回顾一下第2章所解决的问题的特点：先给定一些数据(键盘输入或程序中用赋值语句)，然后按照某个公式计算出一些结果，最后把结果输出到屏幕上，告知用户。这个过程可以说是直线型的，很固定，它们的先后顺序是固定不变的、依次进行的，在这个过程中不需要做任何判断，没有任何智能在里面，对应的程序结构是顺序结构。实际上计算机不仅能计算，按照公式计算，而且还能够有选择地、有判断地采纳不同的计算方案，也就是计算机具有判断决策能力，能像人一样思考。当然，这种能力是人通过程序赋予计算机的。本章要展现的就是在Python中是如何表示判断条件的，是如何做判断决策的。

本章要解决的问题是：

- 让成绩合格的学生通过；
- 按成绩把学生分成两组；
- 按成绩把学生分成多组(百分制和五分制)；
- 判断某年是否为闰年；
- 判断点的位置。

3.1 让成绩合格的学生通过

问题描述：

假设有一个计算机打字训练教室，大一刚入学的同学都要到这个训练教室练习打字。计算机自动考核，成绩60分以上视为合格。训练教室的门口有一个计算机控制的栏杆，它是一个“智能栏杆”，知道每一个参加训练的同学的当前训练成绩，因此当有人走进它时，它

会获取学号，并要求输入成绩，然后计算机会检查输入的成绩是否属实，如果属实并大于或等于 60 分，栏杆将自动打开，允许通过。可想而知，如果成绩小于 60 分智能栏杆会是什么样子。注意，键盘输入成绩的时候必须要诚实，别忘了它是智能栏杆，不然就会有不良记录。写一个程序模拟这个“智能栏杆”。

输入样例 1：

```
What is your grade? 88
```

输出样例 1：

```
Good! You passed!
...
```

输入样例 2：

```
What is your grade? 50
```

输出样例 2：

无

问题分析：

这个问题看起来似乎很复杂，怎么搞一个“智能栏杆”呢？首先，“智能栏杆”知道前来测试的同学的成绩，可以用 input 函数来模拟，即等待同学输入他/她的成绩；控制栏杆的起落可以用一个简单的判断来模拟，即“成绩是大于或等于 60 吗”，如果是，它就升起，允许同学通过，待同学通过之后又落下，等待其他同学的到来。它始终处于监控状态。这里“允许通过”可以用输出一条信息“Good! You passed!”来仿真。因此，整个过程就可以先用 input 获取成绩，获得成绩后，进行一个判断，如果成绩大于或等于 60，打印通过信息；不然什么都不做，继续等待同学输入，这个过程是无限的。图 3.1 是解决这个问题的流程图，下面的算法是解决这个问题的简单版本，没有考虑任何可能的输入错误，而且程序不能停止。

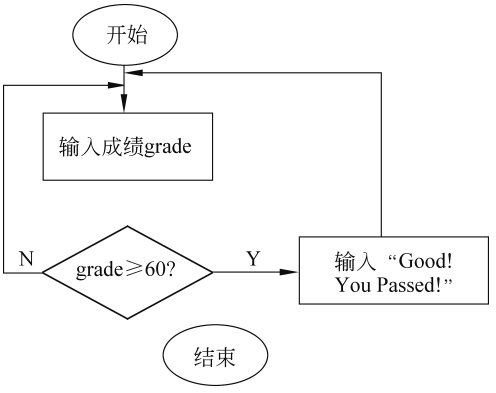

图 3.1　让成绩合格的学生通过的流程图

算法设计：

① 计算机等待输入成绩；

② 如果成绩大于或等于 60，输出“Good! You passed!”，回到①；

③ 否则，回到①。

程序清单 3.1

```
1   #@File: passedsimple.py
2   #@Software: PyCharm
3
```

```
4   while True:
5       grade = int(input("What is your grade? "))
6       if grade >=60:
7           print("Good! You passed!")
```

修改一下上面的算法，让它具有基本的容错能力，当输入非数值数据时提示输入错误重新输入，当输入 Ctrl-Z 或 Ctrl-D(Linux 系统)时程序停止运行，自然流程图也要修改一下，读者可以尝试一下。修改后的实现代码如下：

程序清单 3.1-1

```
1   #@Fle: passedgood.py
2   #@Software: PyCharm
3
4   while True:
5       try:
6           grade = int(input("What is your grade? "))
7           if grade >=60:
8               print("Good! You passed!")
9       except EOFError:          #处理输入 Ctrl-Z/Ctrl-D 时产生的 EOFError 异常
10          break
11      except ValueError:        #处理输入非数值数据时产生的 ValueError 异常
12          print("Your input is not right!")
13          continue
14
15  print("\nBye!")
```

3.1.1 关系运算与逻辑判断

程序清单 3.1 中第 6 行有一个式子 grade >= 60，它是**关系表达式**，其中出现的运算符>=称为**关系运算符**。这个表达式是把 grade 变量引用的对象与 60 进行大于或等于比较，比较的结果有两种可能，或者为真，或者为假，当大于或等于成立时为真，否则为假。由此可见，关系表达式的真假是表示逻辑判断的条件，使用关系表达式就可以让计算机具有一定的“智能”。

Python 支持 6 种关系运算，其中大于>，小于<，大于或等于>=，小于或等于<=，它们与数学上两个数的比较运算>、<、≥、≤相对应，但要注意写法上有所不同。此外，还有等于==，不等于!=两种运算，这与数学上的写法大不相同。这里有两个容易犯的错误，一是把由两个符号构成的关系运算如<=分开书写成< =，中间多了一个空格；二是非常容易把判断相等的关系运算==写成一个=号，这两个都是语法错误。

关系运算像算术四则运算一样，都是**双目运算**，即它们都有两个操作数。由关系运算符连接起来的式子称为**关系表达式**，如 grade >= 60 就是一个关系表达式。

到此为止，已经有三类主要的运算了，它们是算术运算、赋值运算、关系运算。在一个表达式中既可以出现算术运算，也可以出现关系运算，甚至它们的混合运算，因此不同类型的运算符之间必须规定严格的优先级。即使是同一类运算，不同的运算之间也要有严格的结

合性。这三类运算的优先级是：

关系运算的优先级低于算术运算，但高于赋值运算，而关系运算中比较大小的四个运算＞，＜，＞＝，＜＝的优先级又高于判断相等的两个运算＝＝，！＝。

关系运算是左结合的，但一般很少连用，真正使用的时候多加一层括号更清楚。表 3.1 扩展了表 2.2，给出了当前各种运算符的优先级和结合性。

表 3-1　运算的优先级和结合性（优先级从高到低）

运　算　符	含　　义	结　合　性
（）	括号	最近的括号配对
＋，－	单目运算，取正、负	从右向左
＊，/，％	双目运算，乘、除、求余	从左向右
＋，－	双目运算，加、减	从左向右
＞，＜，＞＝，＜＝	双目运算，比较大小	从左向右
＝＝，！＝	双目运算，判断是否相等	从左向右
＝	双目运算，赋值	从右向左

【例 3.1】 设有“a ＝ 1，b ＝ 2，c ＝ 3”，分析下面两个语句中各种运算的顺序：

```
① print("%d\n"%(a+b>c));          //算术运算与关系运算混合
② status = a >b;                  //赋值运算与关系运算混合
```

分析如下：因为算术运算优先于关系运算，所以①的运算顺序为先 a＋b，其结果再与 c 比较；因为关系运算优先于赋值运算，所以②的运算顺序为先 a＞b，结果再赋值给 status。

【例 3.2】 设有“a ＝ 30，b ＝ 20，c ＝ 2”，下面语句正确吗？

```
status = a >b >c;
```

如果正确，status 的值会是多少？

Python 中连续比较运算相当于两个比较之间省略了一个并且运算，即

```
a >b >c  等价于 a >b 并且 b >c
```

其中 30 ＞20 为真，20 ＞ 2 为真，因此 status 结果为真。

3.1.2　逻辑常量与逻辑变量

任何表达式都是有值的，算术表达式的值是算术运算的结果，赋值表达式的值是赋值的结果。Python 中关系表达式的值应该是比较的结果。关系表达式比较的结果只有两种可能，不是真就是假。在计算机内部用 1 表示逻辑真，用 0 表示逻辑假。在 Python 中分别对应逻辑类型 bool 的逻辑常量 True 和 False，这是两个符号常量。例如：

```
>>>a=10
>>>b=20
>>>a>b
False
```

```
>>>a<=b
True
>>>a==b
False
```

Python 语言提供逻辑数据类型 bool，它实际上是整数类型 int 的子类，例如

```
>>>status = a >=b                    >>>print("%d"%status)
>>>atatus                            0
False                                 >>>status = a <b
>>>type(status)                      >>>print("%d"%status)
>>><class 'bool'>                    1
```

或者使用 int 转换器直接输出逻辑常量 True 和 False 的值。

```
>>>print(int(True))                  >>>print(int(False)
1                                     0
```

3.1.3 单分支选择结构

关系表达式的真或假构成了逻辑判断的条件。Python 的选择结构就是通过判断条件的真和假有选择地执行某些语句(分支)。按照分支的多少分为三种选择结构：单分支、双分支和多分支。本节介绍单分支的选择结构。

单分支选择结构用 if 语句表达，具体格式如下：

```
if 判断条件表达式:
    条件为真时执行的语句或语句块
其他语句
```

其中判断条件表达式可以是 3.1.2 节介绍的关系表达式，也可以是其他形式(详见 3.1.4 节)；条件为真时执行的语句可以是任何可执行语句，甚至可以是多个语句构成的**语句块**(也称为**复合语句**)。单分支选择结构的流程图如图 3.2 所示。

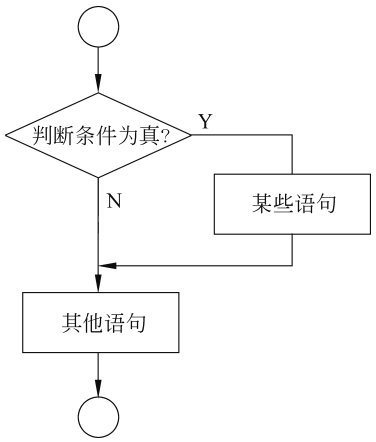

图 3.2 单分支选择结构流程图

注意 if 语句的结构和写法。从结构上来看，可以认为 if 语句有两部分组成，一部分是 if 判断行，if 之后跟一个判断表达式，该表达式不用外加圆括号，另一部分是条件为真时要执行的语句，这一部分可以是另一个单句，也可以是一组语句。但是要注意两点，一是 if 判断行必须以冒号“:”结尾，二是第二部分的语句块必须按缩进格式书写。因为 Python 用缩进格式对语句进行分块，所以缩进的距离要相同。程序清单 3.1 的第 6 行和 7 行合起来才是一个单分支选择结构，或者叫 if 语句。注意它的写法：

```
>>>if grade>=60:
>>>     print("Good! You passed!\n")
```

程序结构通常用**流程图**表示，可以直观地认识它的执行过程，这个 if 语句对应的流程图如图 3.1 或 3.2 所示。在流程图中，一般把判断条件表达式置于菱形框之内，用菱形框表示判断条件，所以通常称菱形框为**判断框**。判断框有且只有一个入口，有且只有一个可以选择的出口，“真”或者“假”。框图中的两个小圆圈表示在其之前或之后是程序的其他部分，它

是连接其他部分的关节，称为**连接框**。从框图可以看出 if 语句的执行过程是：条件为真时执行某个可执行语句或复合语句；条件为假时跳过那个可执行语句或复合语句去执行 if 结构下面的其他语句。

【例 3.3】 对键盘输入的两个整数 a 和 b，输出它们所具有的大小关系。

先简单分析一下。两个数的大小关系可能有多种，大于、小于、大于或等于、小于或等于、不等于、等于，不能只给出其中的一种判断。如果大于关系成立，大于或等于也成立，不等于也成立。输入的数据不同，大小关系也不同。因此需要列出所有可能的大小关系。具体实现见程序清单 3.2。

程序清单 3.2

```
#@File: compare2numbers.py
#@Software: PyCharm
'''
   Enter 2 integer numbers, output the comparing results
'''
a, b = eval(input("Enter two numbers e.g.,2,3"))
if a >b:
    print("%d >%d"%(a,b))
if a >=b :
    print("%d >=%d"%(a,b))
if a == b:
    print("%d == %d"%(a,b))
if a <b:
    print("%d <%d"%(a,b))
if a <=b:
    print("%d <=%d"%(a,b))
if a !=b:
    print("%d !=%d"%(a,b))
```

运行结果：

输入用例 1：

```
2 3
```

输出用例 1：

```
2<3
2<=3
2!=3
```

输入用例 2：

```
3 2
```

输出用例 2：

```
3>2
3>=2
3!=2
```

输入用例 3：

```
3 3
```

输出用例 3：

```
3>=3
3<=3
3==3
```

复合语句是多个语句的复合体,也可称为语句块,在 Python 中通过缩进格式表示复合语句。复合语句本身自成一体,它与程序的其他部分既相互独立又有一定的联系。一个单分支的选择结构,常常包含一个复合语句在内,下面的例子中 9 行到 11 行构成一个复合语句,是条件 grade<60 为真时要执行的语句。还有其他的语句块,你能识别出来吗?

【例 3.4】 写一个程序统计不及格的学生数。

程序清单 3.3

```
1   #@File: nopassed.py
2   #@Software: PyCharm
3
4   nopassed = 0
5   while True:
6       try:
7           grade = eval(input("What is your grade? "))
8           if grade < 60 :
9               print("Sorry! You are not passed!")
10              print("Hope you make great efforts!")
11              nopassed = nopassed +1
12      except EOFError:                    #输入 Ctrl-Z 或 Ctrl-D 时,退出 while
13          break
14      except NameError:                   #如果输入了非数值的字符串,产生这个异常
15          print("Your input is not right!")
16          continue
17
18  print("\nNopassed total:", nopassed)
19  print("Bye!")
```

注意:Nopassed 变量是统计不及格人数的累加变量,它必须初始化为 0。

3.1.4 特殊形式的判断条件

除了利用像 grade >= 60 这样的关系表达式的值作为逻辑判断条件之外,还有一些特殊形式,它们不是关系表达式,但当它们用作判断条件时,系统就会把它们转换为逻辑值。如算术表达式的值、一个整型变量或常量的值、一个字符常量值,甚至是浮点型变量或常量。Python 规定一个表达式的值或某个变量的值或常量,当它们出现在 if 语句或其他含有判断条件的语句中时,只要它们的值非零,就转换为逻辑真,只有它们的值为零时才为逻辑假。简单来说,**非 0 即为逻辑真,0 为逻辑假**。下面分别举例说明。

【例 3.5】 判断一个整数不是偶数(或者是奇数)。

一个数 x 如果它能被 2 整除它就是偶数,也就是说如果 x%2 等于 0,那么 x 就是偶数,那么一个数不是偶数的判定条件为真怎么写呢?答案是 x%2 != 0,即 x 除以 2 其余数不为零,不为零的数当然不等于 0,因此结果为真。下面的 if 语句

```
>>>if x%2 !=0:
>>>    print("%d 不是偶数"%x)
```

按照**非零即为逻辑真**的原则,可以把它简写为

```
>>>if x%2:
>>>    print("%d 不是偶数"%x)
```

这里用算术表达式 x%2 作为逻辑判断条件，当它的值非零时就转化为逻辑真。

【例 3.6】 判断一个整数不是零。

很容易写出“一个整数 x 不是零”为真的条件 x!=0，因此有下面的 if 语句：

```
>>>if x !=0:
>>>    print("%d 不是零"%x)
```

与例 3.4 类似，同样有下面的简写形式

```
>>>if x:
>>>    print("%d 不是零"%x)
```

这里直接使用了变量的值作为判断条件，当 x 不是零时它的值就转换为逻辑真。

现在回头看看 while 1，这个 1 就是一个常量作为逻辑判断条件的例子，1 不为零，所以当然就是逻辑真了。其实，如果把 1 换成任何一个其他不为零的整数，负数也可以，其效果都是一样的，如 while 2，while－100。而且这个常数作为逻辑条件是不变的，所以它永远为真。

还有更多形式的判断条件后面章节会陆续介绍。

3.1.5　比较两个实数的大小

在数学上，比较两个实数与比较两个整数没什么区别，但在计算机中，比较两个实数要特别小心了。由于实数在计算机中存储是有精度的，Python 中的 float 类型的数据只有 16 位或 17 位有效数字，而且可靠的有效位数只有 15 位，所以超出部分的不同就不起作用了。例如：

```
>>>a=3.12345678901234569
>>>b=3.12345678901234561
>>>a==b
True
```

不难看出，虽然浮点型 a 和 b 它们的第 18 个数字不同，在数学上这是两个不相等的实数，但是在 Python 中它们确相等。再如：

```
>>>a=0.123456789012345675
>>>b=0.123456789012345672
>>>a==b
True
```

同样，a 和 b 小数点后第 18 个数字不同，但它们的前 17 位相同，所以 Python 认为它们相等。因此**两个浮点型数是否相等是由它们的精度决定的**。浮点数 float 的默认精度是 16 位到 17 位，在实际问题中未必需要这么高的精度。精度常常用一个小数表示，如 0.01 就是精确到 2 位小数，0.0001 就是精确到 4 位小数，当小数位数更多的时候可以用指数形式表示，如.1e-7 就是精确到第 8 位小数，这种精度数据常常用变量 epsilon(可简写为 eps)表示，即

```
epsilon = .1e-7;
```

Python sys 模块中给出一个 float 型数据的 epsilon，即

```
>>>sys.float_info.epsilon
2.220446049250313e-16
```

这个数对应的普通小数就是

```
0.0000000000000002220446049250313
```

在程序设计的时候**一般不是直接比较两个实数是否相等，而是通过它们之间的误差的精度来判断**，误差如果在允许范围之内就认为是相等的了。标准数学模块 math 中提供了一个求 float 数据的绝对值的函数 fabs(double x)，在程序中用来求两个数的误差(就是相减)的绝对值，如果这个绝对值不超过给定的精度 epsilon，就可以认为这两个数是相等的。我们可以检验一下上面相等的 a 和 b 是否满足这个条件：

```
>>>math.fabs(a-b)<sys.float_info.epsilon
True
```

把 a 和 b 的值修改成

```
>>>a=0.123456789012345685
>>>b=0.123456789012345672
```

或者

```
>>>a=0.123456789012345785
>>>b=0.123456789012345572
```

可以运行>>>math.fabs(a-b)<sys.float_info.epsilon 检验一下。

看一个完整的例子，程序清单 3.4 中给定了一个精度 eps＝0.001 和单精度的 pi＝3.1415926，用户任意输入一个 pi 值存入 yourPi 中，程序通过把 yourPi 与程序中的 pi 值比较，如果它们的误差的绝对值不超过给定的精度 eps，这时就认为 yourPi 符合精度，也可以认为在给定的精度下 yourPi 与程序中的 pi 相同。如果它们的误差的绝对值大于给定的精度，这时认为 yourPi 不符合精度要求。

程序清单 3.4

```
#@File: realNumCompare.py
#@Software: PyCharm

import math
myEpsilon = 0.001
myPi = 3.1415926
yourPi = eval(input("I have a epsilon, Enter your Pi:"))
err = math.fabs(myPi -yourPi)
print("the err is:", err)
if err <=myEpsilon:
    print("Your Pi is ok! Because of the err %7.5f <=myEpsilon %7.5f" \
        %(err,  myEpsilon))
```

```
12  if err >myEpsilon:
13      print("Your Pi is not ok! Because of the err %7.5f >myEpsilon %7.5f"\
            %(err, myEpsilon) )
```

运行结果 1：

```
I have a epsilon, Enter your Pi:3.1415
the err is: 9.259999999988722e-05
Your Pi is ok! Because of the err 0.00009 <=myEpsilon 0.00100
```

运行结果 2：

```
I have a epsilon, Enter your Pi:3.14
the err is: 0.001592599999999944
Your Pi is not ok! Because of the err 0.00159 >myEpsilon 0.00100
```

3.2　按成绩把学生分成两组

问题描述：

教师要把参加某次测验的学生按成绩及格与否分成两组，并统计出各组的人数。

输入样例：

```
88 99 77 66 55 44 -1          //-1 表示输入结束
```

输出样例：

```
you belong in group A
you belong in group A
you belong in group A
you belong in group A
you belong in group B
you belong in group B
aNum = 4
bNum = 2
```

问题分析：

3.1 节的问题只考虑成绩合格者如何处理，成绩不合格者置之不理，即当判断条件为真时去处理事情，而当判断条件为假时就跳过了，没有做任何事情。很多场合我们不仅要描述判断条件为真时做什么，还要对判断条件为假时的情况做出处理。本节的问题仍然是一个判断决策问题。按学生成绩把学生进行分组，就是成绩大于或等于 60 的学生去 A 组，成绩小于 60 的学生去 B 组，并统计出每组的学生数。用简单的单分支选择结构可以解决这个问题吗？回答是可以的。分析下面的程序是否可行，看看存在什么不足。

程序清单 3.5

```
1   #@File: if2parallel.py
2   #@Software: PyCharm
3   aNum = 0
```

```
4   bNum = 0
5   while True:
6       try:
7           grade = eval(input("Enter grades:"))
8           if grade >=60:
9               print("You belong in Group A")
10              aNum = aNum +1
11          if grade <60:
12              print("You belong in Group B")
13              bNum = bNum +1
14      except EOFError:
15          break
16      except NameError:
17          print("Your input is not right!")
18  print()
19  print("aNum = ", aNum)
20  print("bNum = ", bNum)
```

这个实现的正确性是没有问题的,但仔细看看会发现,不管成绩是大于或等于 60,还是小于 60,第 8 行和第 11 行的两个判断都要进行。例如,现在一个学生的成绩是 90,首先经历第 8 行的判断,grade ＞＝ 60 为真,执行第 9 行和第 10 行。紧接着就要执行第 11 行的判断,grade ＜ 60 为假,因此不执行第 12 行和第 13 行。同样如果一个成绩是 50,也要经历同样的两次判断。每个成绩都要判断两次,显然是一种浪费。实际上,成绩大于或等于 60 和成绩小于 60 这两个判断条件之间是紧密相连的,是恰好相反的。如果第一个条件为假,自然就有另一个条件为真,没有必要再去重复判断。对于具有这样性质的判断问题,Python 提供了一种双分支选择结构 **if-else**。下面用双分支的选择结构解决这个问题,其流程图如图 3.3 所示。

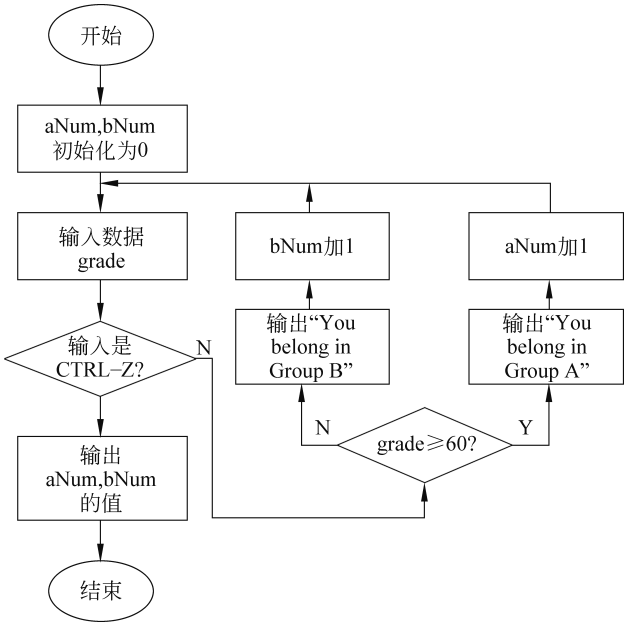

图 3.3　按成绩把学生分成两组的流程图

算法设计：

① 把统计求和变量 aNum，bNum 初始化为 0。

② 输入学生成绩，如果输入了 Ctrl-Z 或 Ctrl-D，执行⑤，否则③。

③ 如果成绩大于或等于 60，输出分到 A 组信息，aNum 加 1，返回②。

④ 否则，输出分到 B 组信息，bNum 加 1，返回②。

⑤ 输出最终统计结果，程序结束。

程序清单 3.6

```
1   #@File: ifelse.py
2   #@Software: PyCharm
3   aNum = 0
4   bNum = 0
5   while True:
6       try:
7           grade = eval(input("Enter grades:"))
8           if grade >=60:
9               print("You belong in Group A")
10              aNum = aNum +1
11          else:
12              print("You belong in Group B")
13              bNum = bNum +1
14      except EOFError:                    #输入了 Ctrl-Z 或 Ctrl-D
15          break
16      except NameError:
17          print("Your input is not right!")
18  print()
19  print("aNum = ", aNum)
20  print("bNum = ", bNum)
```

3.2.1　双分支选择结构

程序清单 3.6 的第 8 行到第 13 行是一个双分支的选择结构，条件 grade＞＝60 为真时执行一个分支，否则执行另一个分支。双分支选择结构用 if-else 语句表示，其一般形式为：

```
if 判断条件表达式:
    条件为真时要执行的语句
else:
    条件为假时要执行的语句
其他语句
```

其中表达式和语句的含义同单分支选择结构一样。它的执行过程如图 3.4 所示，当判断条件为真时执行 if 和 else 之间的语句，否则（隐含着判断条件为

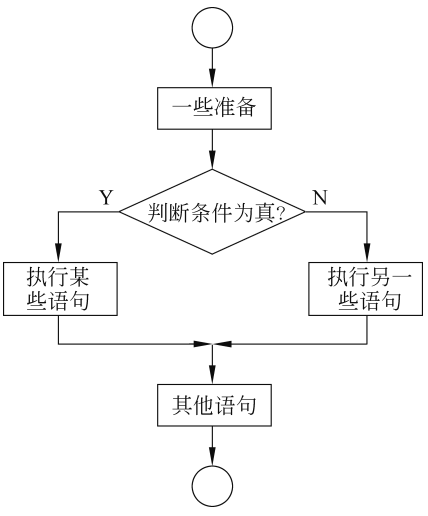

图 3.4　双分支选择结构流程图

假),执行 else 后面的语句。在这个结构中存在两个分支,对于给定数据,只能有一个分支符合判断条件。不管是哪个分支,执行完毕之后都应该执行 if-else 结构下面的“其他语句”。这种双分支选择结构是对称的。

从程序清单 3.5 和程序清单 3.6 的运行结果可以看到,两个平行的单分支选择结构和一个双分支选择结构都能实现本节的问题,其结果完全一致,但是它们的运行过程有很大的不同。两个单分支要判断两次,而一个双分支只判断一次。下面再看几个小例子。

【例 3.7】 判断一个数是奇数还是偶数。

程序实现的片段如下:

```
num = eval(input())
if  num %2 :
    print("num is odd")
else:
    print("num is even")
```

当用户输入一个整数之后,如果输入的是奇数,则 num%2 不为 0,判断条件为真,输出信息 num is odd,如果输入的是偶数,num%2 为 0,判断条件为假,输出 num is even。输出哪条信息(进入哪个分支)由判断条件 num%2 的真假决定。

【例 3.8】 判断一个数是大于或等于零还是小于零。

程序实现的片段如下:

```
num = eval(input())
if  num >=0:
    print("num is equal to 0 or positive number ")
else:
    print("num is a negative number")
```

【例 3.9】 判断一个人的体重是否过大,判断标准是身体指数 t 是否大于 25,其中 $t=w/h^2$(w 为体重,h 为身高)。

程序实现的片段如下:

```
w,h=eval(input())
t = w/(h * h)
if t >25 :
    print("your weight is higher!");
else:
    print("your weight is not higher,\
           but I could not know if your weight is lower!")
```

3.2.2 条件表达式

由于双分支选择结构应用比较频繁,Python 提供了一种特别的运算,称为**条件表达式**,用来表达对称的双分支选择结构,具体格式如下:

```
表达式 1 if 逻辑表达式 2 else 表达式 3
```

这里有三个表达式，两个关键字，可以把它理解成一个三目运算。这个表达式的执行顺序是先执行 if 运算，判断逻辑表达式 2 的真假，如果为真则执行表达式 1，否则执行表达式 3。注意表达式 2 是判断条件，表达式 1 和表达式 3 是对称的两个选择结果。条件表达式的使用非常灵活简洁，通常可以用于有选择的赋值给某个变量。下面看几个例子。

【例 3.10】 求两个数的最大值，对于给定的 a 和 b，

```
max =a if a >b else b
```

max 的值是条件表达式 a if a ＞ b else b 的结果，而条件表达式由逻辑表达 a＞b 的真假决定，当 a＞b 时条件表达的值为 a，否则为 b。

【例 3.11】 用条件表达式判断 print 函数的输出内容是奇数还是偶数。

```
print("num is odd" if  num%2 else "num is even")
```

条件表达式的值是 print 函数的输出值，它由逻辑表达式 num％2 的真假来决定是表达式 1 的字符串 num is odd 还是表达式 2 的字符串 num is even。类似的还有下面的例子。

【例 3.12】 用条件表达式判断 print 函数输出的一个数是正还是负的信息。

```
print("zero or positive" if num >=0 else "negative")
```

【例 3.13】 打印两个数中的较大者，对于给定的 i 和 j。

```
print(i if i >j else j)
```

根据 i＞j 的真假，输出 i 或 j，当 i＞j 时输出 i，否则输出 j。

【例 3.14】 返回两个数中的最大者，对于给定的 i 和 j。

```
return( i if i >j else j)
```

根据 i＞j 的真假，返回 i 或 j，当 i＞j 时返回 i，否则返回 j。

注意：条件表达式一般多用于三个表达式比较简单的情形，并且可以嵌套。

【例 3.15】 条件运算是右结合的，对于给定的 a、b、c、d 条件表达式。

```
a if a >b else c if c>d else d
```

按照右结合的原则，c ＞ d 先作为第二个条件表达式的判断条件，如果它为真取 c，否则取 d，然后再看 a＞b 是否为真，如果为真，则取 a，否则取 c＞d 时条件运算的值。它相当于

```
a if a >b else ( c if c>d else d )
```

3.3　按成绩把学生分成多组（百分制）

问题描述：

写一个程序帮助教师把学生按分数段（90 分以上，80～89 分，70～79 分，60～69 分，小于 60 分）分成多组，并统计各组的人数。

输入样例：

```
please input grades:
```

```
44 55 77 88 99 98 78 67 Ctrl-D
```

输出样例：

```
Failed! group F
Failed! group F
Middle! group C
Better! group B
Good! group A
Good! group A
Middle! group C
Pass! group D
aNum = 2
bNum = 1
cNum = 2
dNum = 1
FNum = 2
```

问题分析：

3.2 节已经使用双分支选择结构解决了把学生按成绩分为两组的问题。本节的问题要求进行更细致的划分，即根据成绩的分数段(90 分以上，80～89 分，70～79 分，60～69 分，小于 60 分)把学生划分为多个组，不妨设为 A、B、C、D、F 组。并分别统计各组的人数。问题中有多个判断条件，而且不是简单的关系表达式所能表达的。90 分以上和 60 分以下比较容易表达，其他几种情况都是一种复合条件，即两个条件要同时满足，如成绩介于 80 分到 89 分之间的复合条件是“成绩大于或等于 80 分”**并且**“成绩小于 90”，在 Python 中允许写成

```
if  80 <=grade <90:
```

这种判断方法中每个分数段用一个 if，采用顺序并列的方式，实现所有分数段的划分。或者把它拆成下面 3.3.1 节将讨论的嵌套的 if 结构来表达。读者可以尝试一下这种判断方法，并分析它有什么不足。

本问题的不同分数段的判断条件实际不是独立的，如 90 分以上的分数段与不是 90 分以上的彼此具有 else 的关系，因此可以用双分支的选择结构，即 if grade＞＝90…else…。而 else 里面还可以有进一步的判断，这是双分支的嵌套。因此按照相邻分数段之间彼此存在的这种联系即可实现，另外本节成绩分数段的判断，不一定必须先判断哪个分数段，后判断哪个，但判断顺序的不同，将导致不同的嵌套顺序。算法设计 1 是按照成绩从大到小的顺序进行判断的，有比较整齐的嵌套描述。即先判断成绩是否大于或等于 90 分，如果不是，再看是否大于或等于 80 分，以此类推。而算法设计 2 是按成绩的客观分布规律考虑判断的顺序，即可能性比较大的先判断。首先判断成绩是否介于 70 分和 80 分之间，如果不是，则有两种可能，一是大于或等于 80 分，二是小于 70 分。如果是大于或等于 80 分，进一步看是否小于 90 分，又分两种情况，小于 90 分和大于或等于 90 分；如果是小于 70 分，进一步看是否大于或等于 60 分，这时又有两种情况，小于和大于或等于 60 分。

算法设计 1：流程图如图 3.5 所示。

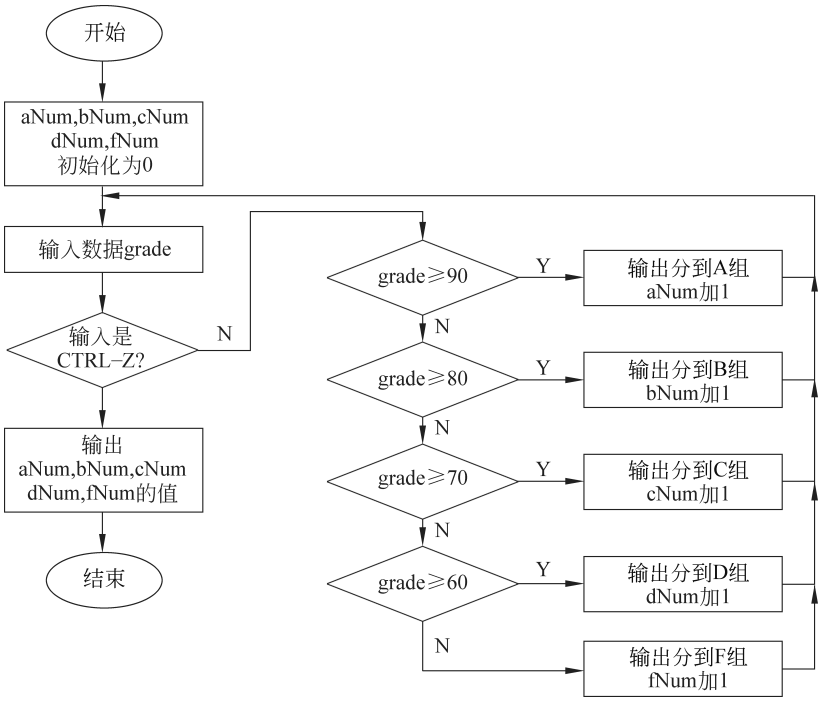

图 3.5　按成绩把学生分成多组算法 1 的流程图

① 把统计求和变量 aNum,bNum,cNum,dNum,fNum 初始化为 0;
② 输入学生成绩;
③ 如果输入没有结束则执行④否则执行⑨;
④ 如果成绩大于或等于 90,输出分到 A 组信息,aNum 加 1,返回到②;
⑤ 否则如果成绩还大于或等于 80,输出分到 B 组信息,bNum 加 1,返回到②;
⑥ 否则如果成绩还大于或等于 70,输出分到 C 组信息,cNum 加 1,返回到②;
⑦ 否则如果成绩还大于或等于 60,输出分到 D 组信息,dNum 加 1,返回到②;
⑧ 否则输出分到 F 组信息,fNum 加 1,返回到②;
⑨ 输出统计结果

程序清单 3.7

```
1   #@File: ifelseif.py
2   #@Software: PyCharm
3
4   aNum = bNum = cNum = dNum = fNum = 0
5
6   while True:
7       try:
8           grade = eval(input("Please enter your grades:"))
9           if grade >=90:
10              print("Good! Group A")
11              aNum = aNum +1
```

```
12          elif grade >=80:
13              print("Better! Group B")
14              bNum = bNum +1
15          elif grade >=70:
16              print("Middle! Group C")
17              cNum = cNum +1
18          elif grade >=60:
19              print("Pass! Group D")
20              dNum = dNum +1
21          else:
22              print("Failed! Group F")
23      except NameError:
24          print("Your input not right!")
25      except EOFError:
26          break
27  print()
28  print("aNum = ", aNum)
29  print("bNum = ", bNum)
30  print("cNum = ", cNum)
31  print("dNum = ", dNum)
32  print("fNum = ", fNum)
```

算法设计 2：流程图如图 3.6 所示。

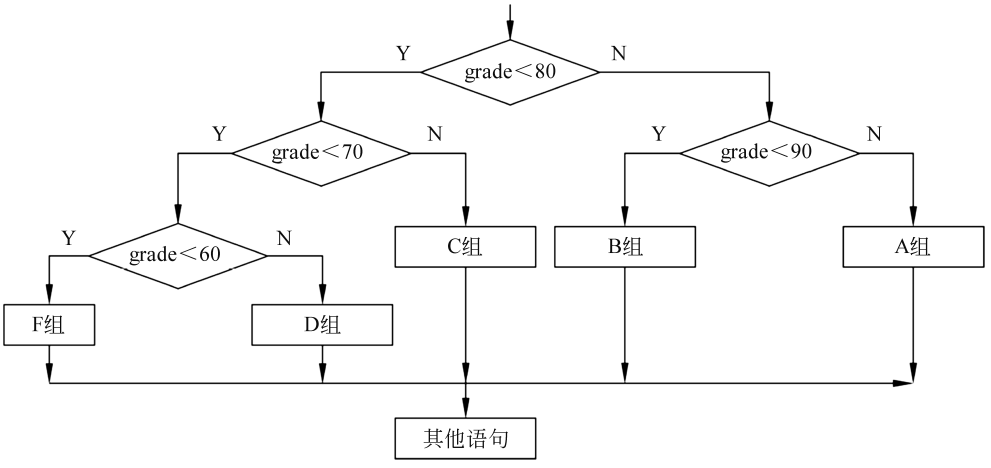

图 3.6　从最可能的成绩开始判断

① 把统计求和变量 aNum,bNum,cNum,dNum,fNum 初始化为 0;

② 输入学生成绩;

③ 如果输入没有结束则执行④,否则执行⑨

④ 如果成绩小于 80 且大于或等于 70,输出分到 C 组信息,cNum 加 1,返回到②;

⑤ 否则如果成绩小于 90 且大于或等于 80,输出分到 B 组信息,bNum 加 1,返回到②;

⑥ 否则(成绩大于或等于 90) 输出或分到 A 组信息,aNum 加 1,返回到②;

⑦ 否则如果成绩小于 70 且大于或等于 60,输出分到 D 组信息,dNum 加 1,返回到②;
⑧ 否则(成绩小于 60),输出分到 F 组信息,fNum 加 1,返回到②;
⑨ 输出统计结果。

程序清单 3.8

```
#@File: ifelseifbetter.py
#@Software: PyCharm

aNum = bNum = cNum = dNum = fNum = 0

while True:
    try:
        grade = eval(input("Please enter your grades:"))
        if grade <80:
            if grade >=70:
                print("Middle! Group C")
                cNum = cNum +1
            elif grade >=60:
                print("Pass! Group D")
                dNum = dNum +1
            else:
                print("Failed! Group F")
                fNum = fNum +1
        elif grade <90:
            print("Better! Group B")
            bNum = bNum +1
        else:
            print("Good! Group A")
            aNum = aNum +1
    except NameError:
        print("Your input is not right!")
    except EOFError:
        break
print()
print("aNum = ", aNum)
print("bNum = ", bNum)
print("cNum = ", cNum)
print("dNum = ", dNum)
print("fNum = ", fNum)
```

3.3.1　嵌套的 if 结构

什么是嵌套呢？简单来说,就是两个东西彼此套在一起,是一种包含关系。一个单分支的 if 能和另一个 if 套在一起吗？让我们仔细分析一下单分支的 if 结构。if 条件为真时要执行的某些语句并没有规定必须是什么语句,当然也可以是另一个 if 结构,即

```
if 表达式 1:
    if 表达式 2:
        表达式 2 为真时执行的语句
```

这样两个 if 结构彼此就嵌在一起了，外层 if 语句的表达式 1 为真时要执行的语句是内层的另一个 if 语句。

这种嵌套结构表达了一种**复合条件**，即如果表达式 1 为真，并且表达式 2 也为真，则执行“表达式 2 为真时要执行的语句”。

【例 3.16】 下面的程序片段表达了什么判断？

```
if grade >=60:
    if grade <70 :
        print("Passed!");
```

其含义是，如果成绩大于或等于 60 分，并且又小于 70 分，计算机回答“Passed!”。这样就实现了判断成绩是否介于 60 和 70 之间。请注意这种嵌套的书写格式，它是逐层缩进的格式，

如果把本节问题中的各个分数段分别进行判断，那么介于 80 和 90 之间的判断就可以用一个独立的 if 嵌套实现，同样介于 70 和 80 之间都可以用一个独立的 if 嵌套实现等，问题便可以得到解决，这种方法的完整实现大家作为一个练习尝试一下。

3.3.2 嵌套的 if-else 结构

与单分支的 if 结构嵌套类似，if-else 结构同样可以嵌套，而且更加灵活方便，其结构如图 3.7 所示。当 if 判断条件为真时要运行的语句或者 else 部分要运行的语句，都可以是另一个 if 或 if-else 结构。双分支的 if-else 是对称结构，if 部分和 else 部分都可以再嵌套其他的选择结构。if-else 结构的 if 部分嵌套另一个 if-else 的基本框架如下：

```
if 表达式 1:
    if 表达式 2:
        表达式 2 为真时执行的语句
    else:
        表达式 2 为假时执行的语句
else:
    表达式 1 为假时执行的语句
```

if-else 结构的 else 部分嵌套另一个 if-else 的基本框架如下：

```
if 表达式 1:
    表达式 1 为真时执行的语句
else:
    if 表达式 2:
        表达式 2 为真时执行的语句
    else:
        表达式 2 为假时执行的语句
```

可以想象，if 部分或 else 部分的 if-else 还可以进一步嵌套其他的选择结构，这样的嵌套

可以包含多层，如图 3.7 所示。不管内部嵌套了多少层，从最外层来看就是一个 if-else 结构。对于本节的分组问题，用 if-else 嵌套实现如下：

```
if grade >=90:
    print("Good! Group A")
    aNum = aNum +1
else:
    if grade >=80:
        print("Better! Group B")
        bNum = bNum +1
    else:
        if grade >=70:
            print("Middle! Group C")
            cNum = cNum +1
        else:
            if grade >=60:
                print("Pass! Group D")
                dNum = dNum +1
            else:
                print("Failed! Group F")
```

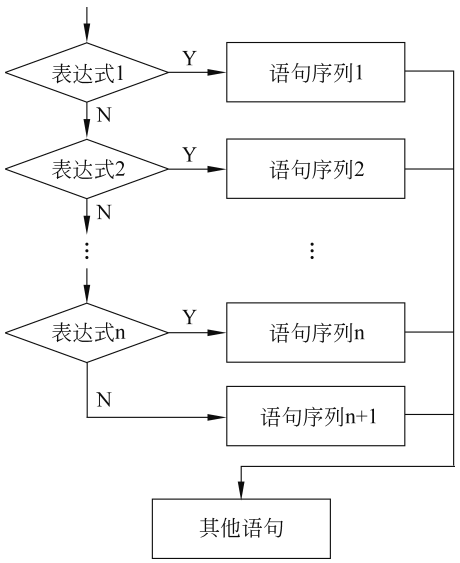

图 3.7　嵌套的 if-else 结构

这个嵌套结构在 grade >= 90 不为真时内嵌了一个另外的 if-else 结构，内嵌的 if-else 进一步判断 grade >= 80 是否为真，当 grade >= 80 为假时又内嵌了一个 if-else 结构，判断 grade >= 70 是否为真，当 grade >= 70 为假时可以继续嵌套另一个 if-else 结构，判断 grade>=60 是否为真，当 grade>=60 为假时都归结为 F 组，嵌套结束。这样的嵌套可能很多层，你认为这种形式的嵌套选择结构有什么不足吗？

还有一点非常值得注意，嵌套的 if-else 结构，虽然层次可以很多，但是 Python 语言的