第 3 章

Python 运算符与流程控制

运算符和流程控制是 Python 程序运算和程序设计的基础，掌握好这两方面知识才可更好地进行 Python 编程。本章首先介绍 Python 中各类运算符的使用规则、注意事项和应用技巧，其次介绍 Python 中的 3 种基本程序结构，最后通过编写代码解决实际问题。

3.1 运算符

什么是运算符？举个简单的例子：1+2=3。例子中，1 和 2 被称为操作数，+被称为运算符。Python 语言支持的运算符非常丰富，包括算术运算符、比较(关系)运算符、赋值运算符、逻辑运算符、位运算符、成员运算符和身份运算符，每种运算符都能完成一种特定的运算。接下来一一学习 Python 的运算符及其用法。

3.1.1 算术运算符

算术运算符用来对数字进行数学运算，Python 中算术运算符有 +、−、*、/、%、**、//。各个算术运算符的含义见表 3-1。

表 3-1 算术运算符

运 算 符	说 明	实 例
+	加，两个对象做加法运算，求和	a+b
−	减，两个对象做减法运算，求差	a−b
*	乘，两个对象做乘法运算，求积	a * b
/	除，两个对象做除法运算，求商	a/b
//	整除，返回商的整数部分(向下取整)	a//b
%	求余，返回除法运算的余数	a%b
**	求次方(乘方)，返回 x 的 y 次幂	a ** b

1. +运算符

当“+”号左右两边的对象是数字时，表示在做加法运算，但是当“+”号一边的对象是字符串，另一边的对象是数字时，需要把数字对象转换成字符串对象，此时“+”号的作用是对字符串进行拼接。例如，下列代码的两个对象做“+”运算时，会出现语法错误：TypeError：

can only concatenate str(not "int") to str,代码如下:

```
#+加法运算符的用法

name = "张三"
age = 18
info = name + "的年龄是"+ age + "岁."
print(info)
```

正确的做法是使用 str(age)方法把整数类型的 age 对象转化成字符串,代码如下:

```
#+加法运算符用法

name = "张三"
age = 18
info = name + "的年龄是"+ str(age) + "岁."
print(info)
```

上述代码的执行结果如图 3-1 所示。

2. -运算符

Python"-"运算符的规则和数学中的减法规则相同。"-"除了可以用作减法运算之外,还可以用作求负运算,如把正数变负数,把负数变正数,代码如下:

```
#-减法运算符的用法

m = 10
m_neg = -m
n = -10.5
n_neg = -n
print("m_neg=",m_neg, ",n_neg=",n_neg)
```

上述代码的执行结果如图 3-2 所示。

3. * 运算符

"*"号除了可以用作乘法运算之外,还可以用来重复显示字符串,也就是将多个相同的字符串连接起来,代码如下:

```
#*乘法运算符的用法

mystr = "你好!"
print(mystr * 3)
```

上述代码的执行结果如图 3-3 所示。

张三的年龄是 18岁。

图 3-1 +运算符的用法

m_neg= -10 ,n_neg= 10.5

图 3-2 -运算符的用法

你好!你好!你好!

图 3-3 * 运算符的用法

4. /和//运算符

Python 中支持两种除法运算符："/"和"//"，其中"/"表示普通除法，使用它计算出来的结果和数学中的除法运算的结果相同，而"//"表示整除，使用它进行计算对结果向下取整，而不是四舍五入，代码如下：

```
#/ 和 //除法运算符的用法

#整数不能除尽
print("21/5 =", 21/5)
print("21//5 =", 21//5)
print("-21//5 =", -21//5)
print("--------------------")

#整数能除尽
print("20/5 =", 20/5)
print("20//5 =", 20//5)
print("-20//5 =", -20//5)
print("--------------------")

#带小数的除法
print("20.5/5 =", 20.5/5)
print("20.5//5 =", 20.5//5)
```

上述代码的执行结果如图 3-4 所示。

```
21/5 = 4.2
21//5 = 4
-21//5 = -5
--------------------
20/5 = 4.0
20//5 = 4
-20//5 = -4
--------------------
20.5/5 = 4.1
20.5//5 = 4.0
```

图 3-4　/和//运算符的用法

从上述代码的执行结果可以发现：不管是否能除尽，也不管参与运算的对象是整数还是小数，/运算符的计算结果总是小数，而//运算符，只有当参与运算的对象中包含小数时，计算结果才是带.0 结果的小数，否则就是整数。

注意：除数不能为 0，因为除以 0 是没有意义的，这将导致 ZeroDivisionError 错误。

5. %运算符

Python 中"%"运算符用来进行模运算，以求得两个数相除的余数，包括整数和小数。Python 使用第 1 个数除以第 2 个数，得到一个整数的商，剩下的值就是余数。如果参与运算的对象中有小数，则求余的结果也是对除数求余后的小数结果，代码如下：

```
# % 求余运算符的用法

print("----- 整数求余 -----")
print("13 % 4 =", 13 % 4)
print("-13 % 4 =", -13 % 4)
print("13 % -4 =", 13 % -4)
print("-13 % -4 =", -13 % -4)
```

```
print("------ 小数求余 ------")
print("12.5 % 1.5  = ", 12.5 % 1.5)
print("-12.5 % 1.5  = ", -12.5 % 1.5)
print("12.5 % -1.5  = ", 12.5 % -1.5)
print("-12.5 % -1.5  = ", -12.5 % -1.5)

print("--- 整数和小数求余 ---")
print("12.5 % 3.5  = ", 12.5 % 3.5)
print("15 % 3.5  = ", 15 % 3.5)
print("-12.5 % 3.5  = ", -12.5 % 3.5)
print("15 % -3.5  = ", 15 % -3.5)
print("-15 % -3.5  = ", -15 % -3.5)
```

```
-----整数求余-----
13%4 = 1
-13%4 = 3
13%-4 = -3
-13%-4 = -1
-----小数求余-----
12.5%1.5 = 0.5
-12.5%1.5 = 1.0
12.5%-1.5 = -1.0
-12.5%-1.5 = -0.5
---整数和小数求余---
12.5%3.5 = 2.0
15%3.5 = 1.0
-12.5%3.5 = 1.5
15%-3.5 = -2.5
-15%-3.5 = -1.0
```

图 3-5　%运算符的用法

上述代码的执行结果如图 3-5 所示。

从上述代码的执行结果可以发现：仅当除数是负数时，求余的结果才是负数，即求余的结果的正负和被除数没有关系，只和除数有关系。“%”运算中当被除数和除数都是整数时，求余的结果也是整数，但是只要有一个数字是小数，求余的结果就是小数。

注意：求余运算的本质是除法运算，所以除数也不能是 0，否则会导致 ZeroDivisionError 错误。

6. ** 运算符

求多个相同因数乘积的运算，叫作次方，也叫乘方，运算的结果叫作幂。在 Python 中用“**”运算符用来求一个数 x 的 y 次方，其中 x 叫作底数，y 叫作指数，可读作 x 的 y 次方或 x 的 y 次幂。由于开方是次方的逆运算，所以也可以使用“**”运算符间接地实现开方运算。

```
# ** 次方(乘方)运算符的用法

print('---- 次方运算 ---- ')
print('2 ** 4  = ', 2 ** 4)
print('2 ** 5  = ', 2 ** 5)
print('---- 开方运算 ---- ')
print('16 ** (1/4)  = ', 16 ** (1/4))
print('32 ** (1/5)  = ', 32 ** (1/5))
```

上述代码的执行结果如图 3-6 所示。

在上述代码中，2 ** 4 表示 2 的 4 次方，执行结果是 16；当用 16 ** (1/4)时，表示对 16 求四分之一次方，结果是浮点数 2.0。

注意：数值才有 ** 运算，对字符串执行 ** 运算会发生错误。

```
----次方运算----
2**4 = 16
2**5 = 32
----开方运算----
16**(1/4) = 2.0
32**(1/5) = 2.0
```

图 3-6　** 运算符的用法

【例 3-1】 从键盘输入两个数，分别对这两个数进行加、减、乘、除、取模、求次方和整除运算，把运算的结果显示在屏幕上，代码如下：

```
#第 3 章/3 - 1.py
#算术运算符综合示例

#从键盘输入
a = int(input("请输入第 1 个数 a = "))
b = int(input("请输入第 2 个数 b = "))

c = a + b
print("这两个数的和: ", c)

c = a - b
print("这两个数的差: ", c )

c = a * b
print("这两个数的积: ", c )

c = a / b
print("这两个数的商: ", c)

c = a % b
print("这两个数取模为", c)

c = a ** b
print("这两个数求幂: ", c)

c = a //b
print("这两个数整除: ", c)
```

上述代码的执行结果如图 3-7 所示。

```
请输入第1个数 a= 21
请输入第2个数 b= 10
这两个数的和:  31
这两个数的差:  11
这两个数的积:  210
这两个数的商:  2.1
这两个数取模为:  1
这两个数求幂:  16679880978201
这两个数整除:  2
```

图 3-7　算术运算符综合示例

3.1.2　比较运算符

比较运算符，也称为关系运算符，用于对两个对象的大小进行比较，这两个对象可以是变量，也可以是常量或表达式的结果。当比较成立时，比较的结果为 True(真)，反之为

False(假)。Python 中的比较运算符为“==、!=、<>、>、<、>=、<=”。各个比较运算符的含义见表 3-2。

表 3-2 比较运算符

运 算 符	说 明	实 例
==	等于,比较两个对象是否相等	x==y
!=	不等于,比较两个对象是否不相等	x!=y
>	大于,返回 x 是否大于 y	x>y
<	小于,返回 x 是否小于 y	x<y
>=	大于或等于,返回 x 是否大于或等于 y	x>=y
<=	小于或等于,返回 x 是否小于或等于 y	x<=y

【例 3-2】 从键盘输入两个对象,分别对这两个对象进行不同的比较判断,并把运算的结果显示在屏幕上,代码如下:

```
#第 3 章/3 - 2.py
#比较运算符综合示例

#从键盘输入
x = input("请输入第 1 个对象 x= ")
y = input("请输入第 2 个对象 y= ")

print("x 等于 y", x == y)
print("x 不等于 y", x != y)
print("x 大于 y", x > y )
print("x 小于 y", x < y)
print("x 大于或等于 y", x >= y)
print("x 小于或等于 y", x <= y)
```

上述代码的执行结果如图 3-8 所示。

若比较的对象不是数字对象而是字符串对象,则通过计算对应位置上每个字符的 Unicode 编码来比较大小,而不是通过字符串的长短来比较。例如对象 abc 和 ab 的比较结果如图 3-9 所示。比较时,从左到右,首先比较两个字符串索引为 0 的字符的 Unicode 编码,被比较的两个字符的大小关系即是两个字符串的大小关系,如果这两个字符相等,则继

```
请输入第1个对象 x= 10
请输入第2个对象 y= 20
x 等于 y False
x 不等于 y True
x 大于 y False
x 小于 y True
x 大于或等于 y False
x 小于或等于 y True
```

图 3-8 比较运算符综合示例(输入对象为数字)

```
请输入第1个对象 x= abc
请输入第2个对象 y= ab
x 等于 y False
x 不等于 y True
x 大于 y True
x 小于 y False
x 大于或等于 y True
x 小于或等于 y False
```

```
请输入第1个对象 x= bc
请输入第2个对象 y= abc
x 等于 y False
x 不等于 y True
x 大于 y True
x 小于 y False
x 大于或等于 y True
x 小于或等于 y False
```

图 3-9 比较运算符综合示例(输入对象为字符串)

续比较后续对应的字符，因为字符串是可迭代对象，先终止迭代的被认为是小的，所以对象abc大于对象ab的运算结果为True。虽然对象bc的长度比对象abc的长度短，但是在对首字母进行比较时，字符b的Unicode编码比字符a的大，因此对象bc比abc大。

注意：＝和＝＝是两种完全不同的运算符，＝是赋值运算符，用来赋值，而＝＝是比较运算符，用来判断左右两边的对象是否相等。

3.1.3　赋值运算符

赋值运算符是把右边的对象传递给左边的变量或常量，右边的对象可以是一个具体的值，也可以是进行某些运算后的结果或者函数调用的返回值。Python中的赋值运算符为“＝、＋＝、－＝、＊＝、/＝、％＝、＊＊＝、//＝”，各个赋值运算符的含义见表3-3。

表3-3　赋值运算符

运　算　符	说　　明	实　　例
＝	简单的赋值运算符	c＝a＋b将a＋b的运算结果赋值给c
＋＝	加法赋值运算符	c＋＝a等效于c＝c＋a
－＝	减法赋值运算符	c－＝a等效于c＝c－a
＊＝	乘法赋值运算符	c＊＝a等效于c＝c＊a
/＝	除法赋值运算符	c/＝a等效于c＝c/a
％＝	取模赋值运算符	c％＝a等效于c＝c％a
＊＊＝	幂赋值运算符	c＊＊＝a等效于c＝c＊＊a
//＝	整除赋值运算符	c//＝a等效于c＝c//a

【例3-3】 从键盘输入两个数值对象，分别对这两个数进行不同的赋值运算，把运算的结果显示在屏幕上，代码如下：

```
#第3章/3-3.py
#赋值运算符综合示例

#从键盘输入
a = int(input("请输入第1个数: a = "))
b = int(input("请输入第2个数: b = "))

c = a + b
print("c = a + b, 则 c = ",c)

c += a
print("c += a, 则 c = ",c)

c -= a
print("c -= a, 则 c = ",c)

c *= a
print("c *= a, 则 c = ",c)
```

```
c /= a
print("c /= a , 则 c = ",c)

c %= a
print("c %= a, 则 c =",c)

c **= a
print("c **= a, 则 c =",c)

c //= a
print("c //= a, 则 c =",c)
```

上述代码的执行结果如图 3-10 所示。

```
请输入第1个数: a = 4
请输入第2个数: b = 10
c = a + b , 则 c =  14
c += a , 则 c =  18
c -= a , 则 c =  14
c *= a , 则 c =  56
c /= a , 则 c =  14.0
c %= a, 则 c = 2.0
c **= a, 则 c = 16.0
c //= a, 则 c = 4.0
```

图 3-10　赋值运算符综合示例

在上述代码中,算式 c+=a,+=使用了加法赋值运算符,它的效果等同于 c=c+a,先把对象 c 和 a 做加法运算,再把结果赋值给对象 c。其他赋值运算符同理。

Python 中的赋值表达式也是有值的,整个表达式的值就是左边变量的值,因此,如果继续将赋值表达式再赋值给另外一个变量,这就构成了连续赋值,示例如下:

```
a = b = c = 50
```

由于赋值运算符“=”具有右结合性,整个表达式的赋值顺序为 c=50,表示将 50 赋值给 c,所以 c 的值是 50,并且 c=50 这个表达式的值也是 50。b=c=50 表示将 c=50 的值赋给 b,因此 b 的值也是 50。以此类推,a 的值也是 50。最终 a、b、c 3 个变量的值都是 50。

注意:c+=100 等价于 c=c+100,这种赋值运算符只能针对已经存在的变量赋值,如果 c 没有提前定义,这种写法就是错误的,因为它的值是未知的。

3.1.4　位运算符

位运算符用来进行二进制计算,因此位运算只能操作整数。Python 中位运算符为“&、|、^、~、<<、>>”。各个位运算符见表 3-4。

表 3-4　位运算符

运算符	说　明	实例
&	按位与运算符:参与运算的两个二进制位,如果都为 1,则该位的结果为 1,否则为 0	a&b
|	按位或运算符:参与运算的两个二进制位,如果有一个为 1,结果位就为 1	a|b
^	按位异或运算符:参与运算的两个二进制位,相同时,结果为 0,相异时,结果为 1	a^b
~	按位取反运算符:对运算数的每个二进制位取反,即把 1 变为 0,把 0 变为 1。~x 类似于-x-1	~a

续表

运算符	说　明	实例
<<	按位左移运算符：运算数的各个二进制位全部左移若干位，由<<右边的数字指定移动的位数，高位丢弃，低位补 0	a << b
>>	按位右移运算符：运算数的各个二进制位全部右移若干位，由>>右边的数字指定移动的位数	a >> b

1. &运算符

按位与运算符“&”是双目运算符。两个操作数 x、y 按相同位置的二进制位进行与操作，当两个位置上都是 1 时，位的与结果为 1，否则为 0。规则如下：

```
#&按位与运算符的用法

1&1 = 1
1&0 = 0
0&1 = 0
0&0 = 0
```

当 a=49，b=23 时，c=a&b，表示把 49 的二进制 0011 0001 和 23 的二进制 0001 0111 按位做与运算，结果为 0001 0001，转化成十进制为 17，赋值给 c。

2. |运算符

按位或运算符“|”是双目运算符。两个操作数 x、y 按相同位置的二进制位进行或操作，只要有一个位置是 1，其结果为 1，否则为 0。规则如下：

```
#|按位或运算符的用法

1|1 = 1
1|0 = 1
0|1 = 1
0|0 = 0
```

当 a=49，b=23 时，c=a|b，表示把 49 的二进制 0011 0001 和 23 的二进制 0001 0111 按位做或运算，结果为 0011 0111，转化成十进制为 55。

3. ^运算符

按位异或运算符“^”是双目运算符。两个操作数 x、y 按相同位置的二进制位进行异或操作，当位置上的数相同时结果为 0，否则为 1。规则如下：

```
#^按位异或运算符的用法

1^1 = 0
1^0 = 1
0^1 = 1
0^0 = 0
```

当 a=49,b=23 时,c=a^b,表示把 49 的二进制 0011 0001 和 23 的二进制 0001 0111 按位做异或运算,结果为 0010 0110,转化成十进制为 38。

4. ～运算符

按位取反运算符“～”是单目运算符。进行按位取反时,把操作数的二进制的每位都取反。可参照公式～x,类似于－x－1,因此当 a=49 时,～a 的值为－50。

5. <<运算符

两个操作数 x、y,将 x 按二进制形式向左移动 y 位,末尾补 0,符号位保持不变。向左移动一位等同于乘以 2。例如 1 的八位二进制数是 0000 0001 现在向左移动两位(1 << 2),结果为 0000 0100,转化成十进制为 4。当 a=49 时,a << 2,表示把 49 的二进制 0011 0001 向左移动 2 位,结果为 1100 0100,转化成十进制为 196。

6. >>运算符

两个操作数 x、y,将 x 按二进制形式向右移动 y 位,符号位保持不变。向右移动一位等同于除以 2。将操作数的二进制的所有位向右移动指定的位数。例如 10 的二进制数是 0000 1010 现在向右移动两位(10 >> 2),结果为 0000 0010,即被挤走了最后的两位,转化成十进制为 2。当 a=49 时,a >> 2,表示把 49 的二进制 0011 0001 向左移动 2 位,结果为 0000 1100,转化成十进制为 12。

【例 3-4】 从键盘输入两个数字对象,对这两个数分别进行不同的位运算,把运算的结果显示在屏幕上,代码如下:

```
#第 3 章/3 - 4.py
#位运算符综合示例

#从键盘输入
a = int(input("请输入第 1 个数: a = "))
b = int(input("请输入第 2 个数: b = "))

c = a & b          #0b10001 = 17
print("c = a & b , 则 c = ",c)

c = a | b          #0b110111 = 55
print("c = a | b , 则 c = ",c)

c = a ^ b          #0b100110 = 38
print("c = a ^ b , 则 c = ",c)

c = ～a            # - 0b110010 = - 50
print("c = ～a , 则 c = ",c)

c = a << 2         #0b11000100 = 196
print("c = a << 2 , 则 c = ",c)

c = a >> 2         #0b1100 = 12
print("c = a >> 2, 则 c = ",c)
```

上述代码的执行结果如图 3-11 所示。

```
请输入第1个数: a = 49
请输入第2个数: b = 23
c = a & b , 则 c =  17
c = a | b , 则 c =  55
c = a ^ b , 则 c =  38
c = ~a , 则 c =  -50
c = a << 2 , 则 c =  196
c = a >> 2, 则 c = 12
```

图 3-11　位运算符综合示例

3.1.5　逻辑运算符

Python 逻辑运算符是用来进行逻辑判断的运算符，它可以操作任何类型的表达式，不管表达式是不是布尔类型；同时，逻辑运算的结果也不一定是布尔类型，它也可以是任意类型。逻辑运算符为“and、or、not”。各个逻辑运算符的含义见表 3-5。

表 3-5　逻辑运算符

运　算　符	说　　明	实　　例
and	逻辑与运算，等价于数学中的“且”	x and y
or	逻辑或运算，等价于数学中的“或”	x or y
not	逻辑非运算，等价于数学中的“非”	not x

1. and 运算符

假设 x、y 为两个表达式，x and y 表示当 x 和 y 两个表达式都为真时，x and y 的结果才为真，否则为假。

```
#and 逻辑与运算符

print(3 > 2 and 4 > 3)          #输出 True
print(3 < 2 and 4 > 3)          #输出 False
print(3 and 4 > 3)              #输出 True
print(3 and 4 < 3)              #输出 False
print(1 and 2)                  #输出 2
print(2 and 1)                  #输出 1
print(0 and 1)                  #输出 0
print(1 and 0)                  #输出 0
```

当 and 左右两边都是表达式时，例如 print(3 > 2 and 4 > 3)，and 左边表达式的结果为真，and 右边表达式的结果也为真，因此 print(3 > 2 and 4 > 3)的输出结果为 True，而对于语句 print(3 < 2 and 4 > 3)，and 左边表达式的结果为假，and 右边表达式的结果为真，因此 print(3 < 2 and 4 > 3)的输出结果为 False。

当 and 一边是变量，另一边是表达式时，例如 print(3 and 4 > 3)，由于 Python 中非 0 数字被当作 True 处理，所以 and 的左边为真，and 右边表达式的结果也为真，因此 print(3 and

4 > 3)的输出结果为 True,而对于语句 print(3 and 4 < 3),and 右边表达式的结果为假,因此 print(3 and 4 < 3)的输出结果为 False。

当 and 两边都是变量而非表达式时,如果 and 两边都是非 0 数,则结果输出 and 右边的变量值,例如 print(1 and 2),输出结果为 2,print(2 and 1),输出结果为 1,而如果 and 有一边的值为 0,则结果就是 0,例如 print(0 and 1)和 print(1 and 0)的输出结果都为 0。

2. or 运算符

假设 x、y 为两个表达式,x or y 表示当 x、y 两个表达式只要一个为真,运算的结果就为真,当两个都为假时,运算结果为假。

```
#or 逻辑与运算符

print(3 > 2 or 4 < 3)                          #输出 True
print(3 < 2 or 4 > 3)                          #输出 True
print(3 < 2 or 4 < 3)                          #输出 False
print(3 or 4 > 3)                              #输出 3
print(4 < 3 or 3)                              #输出 3
print(4 > 3 or 3)                              #输出 True
print(1 or 2)                                  #输出 1
print(2 or 1)                                  #输出 2
print(0 or 1)                                  #输出 1
print(1 or 0)                                  #输出 1
print("" or "https://www.icourse163.org/")     #输出 https://www.icourse163.org/
```

当 or 左右两边都是表达式时,例如 print(3 > 2 or 4 < 3),or 左边表达式的结果为真,因此 print(3 > 2 or 4 > 3)的输出结果为 True;对于语句 print(3 < 2 or 4 > 3),or 右边表达式的结果为真,因此 print(3 > 2 or 4 > 3)的输出结果为 True,而对于语句 print(3 < 2 or 4 < 3),or 左右两边表达式的结果都为假,因此 print(3 < 2 or 4 < 3)的输出结果为 False。

当 or 一边是变量,另一边是表达式时,例如 print(3 or 4 > 3),由于 or 的左边为真,因此 3 or 4 > 3 的结果取左边的值,print(3 or 4 > 3)的输出结果为 3;对于语句 print(4 < 3 or 3),由于 or 左边表达式为假,右边为真,因此 print(4 < 3 or 3)的输出结果为 or 右边的值 3,而对于语句 print(4 > 3 or 3),or 左边表达式的结果为 True,因此 print(4 > 3 or 3)的输出结果为左边表达式的结果,因此输出 True。

当 or 两边都是变量而非表达式时,如果 or 两边都是非 0 数值,则结果输出 or 左边的变量值,例如 print(1 or 2),输出结果为 1,print(2 or 1),输出结果为 2,而如果 or 有一边的值为 0,则结果是输出另一边非 0 值,例如 print(0 or 1)和 print(1 or 0),输出结果都为 1。在 Python 中,以下变量都会被当成 False:任何数值类型的 0、""或空字符串、空元组()、空列表[]、空字典{}等,因此语句 print("" or "https://www.icourse163.org/")中,or 左边的结果为 False,因此语句输出结果为 or 右边的值 https://www.icourse163.org/。

3. not 运算符

假设 x 为表达式,当 x 为真时,not x 运算的结果就为假;当 x 为假时,not x 运算的结

果为真，代码如下：

```
#not 逻辑非运算符

print(not(2 > 1))          #输出 False
print(not(1 > 2))          #输出 True
print(not 1)               #输出 False
print(not 0)               #输出 True
print(not True)            #输出 False
print(not False)           #输出 True
```

当 not 右边是表达式时，如果表达式的结果为真，如 print(not(2 > 1))，则输出结果为 False；如果表达式的结果为假，如 print(not(1 > 2))，则输出结果为 True；由于 1 是非 0 数，1 当作 True 看待，因此 print(not 1)的输出结果为 False，而 print(not 0)的输出结果为 True。

在 Python 中，and 和 or 不一定会计算右边表达式的值，有时只计算左边表达式的值就可以得到最终结果。

【例 3-5】 执行以下逻辑运算符的代码，输出结果显示在屏幕上，代码如下：

```
#第 3 章/3 - 5.py
#逻辑运算符综合示例

a = 10
print("---- False and xxx -----")
print( False and(a: = 20) )
print(a)
print("---- True and xxx -----")
print( True and(a: = 30) )
print(a)
print("---- False or xxx -----")
print( False or(a: = 40) )
print(a)
print("---- True or xxx 形式 -----")
print( True or(a: = 50) )
print(a)
```

上述代码的执行结果如图 3-12 所示。

```
----False and xxx-----
False
10
----True and xxx-----
30
30
----False or xxx-----
40
40
----True or xxx 形式-----
True
40
```

图 3-12　逻辑运算符综合示例

如果 and 左边表达式的值为假，就不用计算右边表达式的值了，因为不管右边表达式的值是什么都不会影响最终结果，最终结果都是假，此时 and 会把左边表达式的值作为最终结果。在以上代码中，语句“False and(a:＝20)”的 and 左边值为假，不需要再执行右边的表达式了，所以 and 右边的赋值并没有执行，因此 a 的输出值为 10。

如果左边表达式的值为真，则最终值是不能确定的，and 会继续计算右边表达式的值，并将右边表达式的值作

为最终结果。在以上代码中,语句“True and(a:=30)”的 and 左边的值为真,还需要执行右边的表达式才可以得到最终的结果,因此会执行 a:=30 语句,a 的输出值为 30。

后面的代码与此类似。

注意:**:=是 Python 3.8 后才具有的新特性,称为海象运算符。它将值赋给变量,整体又作为表达式的一部分,使代码更加简洁。**

3.1.6 成员运算符

除了以上的一些运算符之外,Python 还支持成员运算符。成员运算符为“in、not in”。各个成员运算符的描述见表 3-6。

表 3-6 成员运算符

运算符	说 明	实 例
in	如果在指定的序列中找到值,则返回 True,否则返回 False	x in y
not in	如果在指定的序列中没有找到值,则返回 True,否则返回 False	x not in y

1. in 运算符

如果在指定的序列中找到元素,就会返回布尔值 True;如果在指定的序列中找不到元素,就会返回布尔值 False。

```
#in 包含运算符的用法

list = [1,2,3,4,5]
a = 6
print("a in list 的结果:",a in list)
string = 'Hello'
b = 'e'
print("b in string 的结果:",b in string)
```

上述代码的执行结果如图 3-13 所示。

```
a in list 的结果: False
b in string 的结果: True
```

图 3-13 in 包含运算符的用法

列表 list 中不包含 a,因此 a in list 的运算结果为 False,字符串 string 中包含 b,因此 b in string 的运算结果为 True。

2. not in 运算符

如果在指定的序列中找到元素,就会返回布尔值 False;如果在指定的序列中找不到元素,就会返回布尔值 True。

```
#not in 非包含运算符的用法

list = [1,2,3,4,5]
a = 6
string = 'Hello'
```

```
b = 'e'
print("a not in list 的结果: ",a not in list)
print("b not in string 的结果: ",b not in string)
```

上述代码的执行结果如图 3-14 所示。

```
a not in list 的结果:  True
b not in string 的结果:  False
```

图 3-14　not in 非包含运算符的用法

列表 list 中不包含 a，因此 a not in list 的运算结果为 True，字符串 string 中包含 b，因此 b not in string 的运算结果为 False。

3.1.7　身份运算符

身份运算符用于比较两个对象的存储单元，是判断它们是否相同的一种运算符号。Python 中，身份运算符只有 is 和 is not 两种，返回的结果是布尔类型，其描述见表 3-7。

表 3-7　身份运算符

运　算　符	说　　明	实　　例
is	is 用于判断两个标识符是不是引用自同一个对象	x is y
is not	is not 用于判断两个标识符是不是引用自不同对象	x is not y

1. is 运算符

如果两个标识符引用的对象一致，就会返回布尔值 True；如果两个标识符引用的对象不一致，就会返回布尔值 False。is 身份运算符的用法，代码如下：

```
#is 身份运算符用法

a = [1,2,3]
b = [2 - 1,3 - 1,4 - 1]
c = a
print("a 的值: ",a)
print("b 的值: ",b)
print("c 的值: ",c)
print("a is b 的结果: ",a is b)
print("a is c 的结果: ",a is c)
```

```
a的值:  [1, 2, 3]
b的值:  [1, 2, 3]
c的值:  [1, 2, 3]
a is b 的结果:  False
a is c 的结果:  True
```

图 3-15　is 身份运算符用法

上述代码的执行结果如图 3-15 所示。

通过代码的执行结果可发现，a 和 b 引用的对象是不一致的，所以 a is b 输出的结果是 False，因为变量 b 是需要计算的，虽然计算之后得到的列表看起来跟变量 a 一模一样，但是计算之前的过程每个元素是要存储的，变量 a 当中的元素都是数字，计算机直接存储结果，而变量 b 当中的每个元素都是表达式，表达式的存储跟单个元素的存储是不一致的，列表、元组都是如此，而变量 c 被赋值为 a，所以变量 c 和变量 a 引用的是同一块存储空间，a is c 输出的结果是 True。

2. is not 运算符

如果两个标识符引用的对象一致,就会返回布尔值 False;如果两个标识符引用的对象不一致,就会返回布尔值 True。is not 身份运算符的用法,代码如下:

```
#is not 身份运算符用法

a=[1,2,3]
b=[2-1,3-1,4-1]
c=a
print("a 的值:",a)
print("b 的值:",b)
print("c 的值:",c)
print("a is not b 的结果:",a is not b)
print("a is not c 的结果:",a is not c)
```

上述代码的执行结果如图 3-16 所示。

变量 a 和 b 引用的对象是不一致的,因此 a is not b 的运算结果为 True;a 和 c 引用的对象是一致的,因此 a is not c 的运算结果为 False。

特别注意:is 与==的区别。

is 用于判断两个变量的引用对象是否为同一块内存空间,而==用于判断引用变量的值是否相等,代码如下:

```
a = [1, 2, 3]
b = a
print("b is a 的结果:",b is a)
print("b == a 的结果:",b == a)
b = a[:]
print("b is a 的结果:",b is a)
print("b == a 的结果:",b == a)
```

上述代码的执行结果如图 3-17 所示。

```
a的值: [1, 2, 3]
b的值: [1, 2, 3]
c的值: [1, 2, 3]
a is not b 的结果: True
a is not c 的结果: False
```

图 3-16 is not 身份运算符

```
b is a的结果: True
b == a的结果: True
b is a的结果: False
b == a的结果: True
```

图 3-17 is 和==的区别

b=a 表示为变量 b 赋值 a,此时 a 和 b 都指向同一对象,所以 b is a 和 b==a 的结果都为 True。当改动变量 a 或者 b 的值时,另一个也会跟着改动。

a[:]是在对列表 a 进行切片,切片的结果为[1,2,3],b=a[:]表示把切片结果对象[1,2,3]赋值给 b,此时 b 和 a 指向的是不同的对象,所以 b is a 的结果为 False,当改动变量 a 或者 b 的值时,不会影响另一个变量,但是 a 和 b 的值都为[1,2,3],因此 b==a 的结果为 True。

3.1.8　运算符优先级


4min

Python 中常用的运算符在混合使用时，要注意优先级，当一个表达式中出现多个运算符时，Python 会先比较各个运算符的优先级，从而决定表达式的执行顺序。运算符的优先级见表 3-8。表中列出了从最高优先级到最低优先级的常用运算符，其中指数运算符的优先级最高，赋值运算符的优先级最低。

表 3-8　Python 运算符优先级

运　算　符	说　　明	结　合　性
**	指数运算符的优先级最高	右
～、+、-	按位取反运算符，正数/负数运算符	右
*、/、%、//	乘、除、取模和整除	左
+、-	加法、减法运算符	左
>>、<<	按位右移、按位左移运算符	左
&	按位与运算符	右
^	按位异或运算符	左
|	按位或运算符	左
<=、<、>、>=、!=、==	比较运算符	左
is、is not	身份运算符	左
in、not in	成员运算符	左
not	逻辑非运算符	右
and	逻辑与运算符	左
or	逻辑或运算符	左
=、%=、/=、//=、-=、+=、*=、**=	赋值运算符	右

【例 3-6】 执行以下运算符的代码，输出结果显示在屏幕上，代码如下：

```
#第 3 章/3-6.py
#运算符优先级

print(3 == 2 or 4 > 2)          # == 和>符号的优先级高于 or
print(5 >= 2 and 4 > 3)         #>= 和>的优先级高于 and
print(4 + 4 << 2)               # + 的优先级高于<<
print(100 //25 * 5)             #//和 * 的优先级相同
print(1 or 2 and 3)             #and 优先级比 or 要高
print(-3 ** 2)                  # ** 优先级高于 -
```

上述代码的执行结果如图 3-18 所示。

==和>符号的优先级高于 or，第 1 条打印语句 print(3==2 or 4>2)，先计算 3==2 和 4>2，结果分别为 False 和 True，最后表达式等价于 False or True，因此最终输出结果为 True。

```
True
True
32
20
1
-9
```

图 3-18　运算符优先级综合示例

>=和>的优先级高于 and,第 2 条打印语句 print(5 >=2 and 4 > 3),先计算 5 >=2 和 4 > 3,结果都为 True,最后表达式等价于 True and True,因此最终输出结果为 True。

+的优先级高于<<,第 3 条打印语句 print(4+4 << 2),先计算 4+4,得到结果 8,再执行 8 << 2,得到结果 32,因此最终输出的结果为 32。像这种不好确定优先级的表达式,可以给子表达式加上(),也就是写成(4+4)<< 2,这样看起来就一目了然了,不容易引起误解。当然,也可以使用()改变程序的执行顺序,例如 4+(4 << 2),先执行 4 << 2,得到结果 16,再执行 4+16,得到结果 20。

//和 * 的优先级相同,此时不能只依赖运算符的优先级了,还要参考运算符的结合性。//和 * 都具有左结合性,因此先执行左边的除法,再执行右边的乘法,因此第 4 条 print(100//25 * 5)的最终结果是 20。

and 优先级高于 or,第 5 条打印语句 print(1 or 2 and 3),先计算 2 and 3,得到结果 3,再执行 1 or 3,结果为 1,因此最终输出结果为 1。

** 优先级高于-,第 6 条打印语句 print(-3 ** 2),先计算 3 ** 2,得到结果 9,再和-结合,因此最终输出结果为-9。

最后,对于运算符的优先级,需要注意以下几点:

(1) 只有在优先级相同的情况下才会考虑结合性,结合性决定先执行哪个运算符:如果是左结合性就先执行左边的运算符,如果是右结合性就先执行右边的运算符。

(2) 不要把一个表达式写得过于复杂,如果一个表达式过于复杂,则可以尝试把它拆分。

(3) 不要过多地依赖运算符的优先级来控制表达式的执行顺序,这样的表达式可读性太差,应尽量使用括号()来控制表达式的执行顺序。


11min

3.2 顺序结构

按照程序执行流程划分,Python 程序设计中的 3 种基本结构是顺序结构、选择结构和循环结构。顺序结构是让程序按照语句的顺序,从头到尾依次执行每条 Python 代码,不重复执行任何代码,也不跳过任何代码。它是流程控制中最简单的一种结构,也是最基本的一种结构。模块导入语句、赋值语句、输入/输出语句等都是顺序结构语句。顺序结构的流程图如图 3-19 所示。

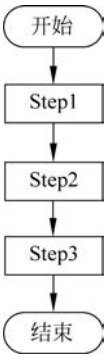

图 3-19 顺序结构流程图

顺序结构的程序特点是语句必须按照从上到下的方向执行,即根据箭头所指的方向依次执行,不能跳过其中某一条语句不执行,也不可一条语句执行多次。

【例 3-7】 输入圆的半径,计算圆的周长和面积,输出结果显示在屏幕上,代码如下:

```
#第 3 章/3 - 7.py
#顺序结构

import math
radius = float(input("请输入圆的半径(cm): "))
circumference = 2 * math.pi * radius
area = math.pi * radius * radius
print("圆的周长为: %.2f\n 圆的面积为: %.2f" %(circumference, area))
```

上述代码的执行结果如图 3-20 所示。

```
请输入圆的半径(cm): 2.5
圆的周长为:15.71
圆的面积为:19.63
```

图 3-20　顺序结构示例

在上述代码中，首先输入圆的半径，根据半径求圆的周长和面积，最后打印输出。程序语句按照从上到下的顺序依次执行，没有跳过任何一行代码。

3.3　选择结构

选择结构也叫分支结构，是指在程序运行过程中通过对条件的判断，根据条件是否成立而选择不同流向的算法结构，它通常是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。选择结构根据分支的多少，分为单分支选择结构、双分支选择结构和多分支选择结构。根据实际需要，还可以在一个选择结构中嵌入另一个选择结构。

3.3.1　单分支选择

单分支选择结构用于处理单个条件、单个分支的情况，可以用 if 语句实现，其语法格式如下：

```
if 表达式:
    执行语句块……
```

if 后面的表达式表示条件，其结果为布尔值，在该表达式后面必须加上冒号。语句块可以是单个语句，也可以是多条语句。语句块必须向右缩进，如果包含多条语句，则这些语句必须具有相同的缩进量。如果语句块中只有一个语句，则语句块可以和 if 语句写在同一行上，即在冒号后面直接写出条件成立时要执行的语句。

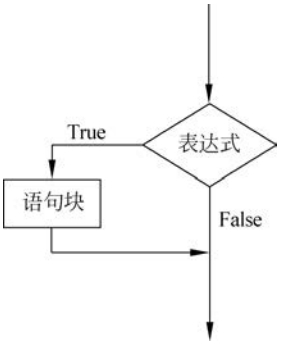

图 3-21　单分支选择结构流程图

单分支选择结构的流程图如图 3-21 所示。

由图 3-21 可以看出，单分支选择结构的执行流程是：首先计算 if 后面表达式的值，如果该值为 True，则执行语句块，然后执行 if 语句的后续语句；如果该值为 False，则跳过语句块，直接执行 if 语句的后续语句。

【例 3-8】 从键盘输入两个数，比较其大小，把较大的值显示输出，代码如下：

```
#第 3 章/3 - 8.py
#单分支选择结构

#从键盘输入
x = int(input("请输入第 1 个数: "))
y = int(input("请输入第 2 个数: "))
m = x
if m < y:          #如果 m < y 成立
   m = y
print("两个数的较大者是: ",m)
```

上述代码的执行结果如图 3-22 所示。

```
请输入第 1 个数: 10
请输入第 2 个数: 20
两个数的较大者是:  20
```

图 3-22　单分支选择结构求两数的较大者

在上述代码中，首先从键盘输入两个数并分别赋给 x 和 y，然后把第 1 个数 x 赋给临时变量 m，接着比较 m 和第 2 个数 y 的大小，如果 m 的值小于 y，则把 y 的值赋给 m，也就是说，如果 m 的值大于或等于 y，则会跳过语句 m＝y，因此，m 的值一直是两数中的较大者。

3.3.2　双分支选择

双分支选择结构用于处理单个条件、两个分支的情况，可以用 if-else 语句实现，其语法格式如下：

```
if 表达式:
    执行语句块 1……
else:
    执行语句块 2……
```

if 后面表达式表示条件，其结果为布尔值，在该表达式后面必须加上冒号。语句块 1 和语句块 2 都可以是单个语句或多个语句，这些语句块必须向右缩进，而且语句块中包含的各个语句必须具有相同的缩进量。

双分支选择结构的流程图如图 3-23 所示。

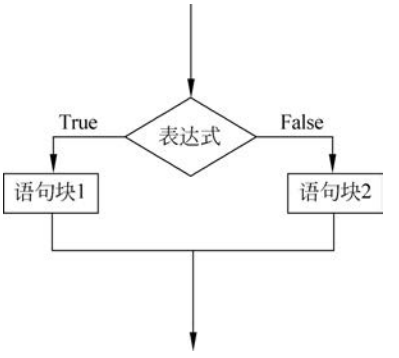

图 3-23　双分支选择结构流程图

由图 3-23 可以看出，if-else 语句的执行流程如下：首先计算 if 后面表达式的值，如果该值为 True，则执行语句块 1，否则执行语句块 2；执行语句块 1 或语句块 2 后接着执行 if-else 语句的后续语句。

【例 3-9】　飞行员的报考条件是年龄在 18 岁至 30 岁之间，并且身高在 170cm 至 185cm 之间。编写一个程序，从键盘输入年龄和身高，判断是否符合飞行员的报考条件，代码如下：

```
#第 3 章/3 - 9.py
#双分支选择结构
```

```
#从键盘输入
age = int(input("请输入年龄: "))
height = int(input("请输入身高: "))
if age >= 18 and age <= 30 and height >= 170 and height <= 185:
   print("恭喜,你符合报考飞行员的条件")
else:
   print("抱歉,你不符合报考飞行员的条件")
```

上述代码的执行结果如图 3-24 所示。

```
请输入年龄: 18                    请输入年龄: 30
请输入身高: 175                   请输入身高: 165
恭喜，你符合报考飞行员的条件      抱歉，你不符合报考飞行员的条件
```

图 3-24　双分支选择结构判断飞行员是否符合报考条件

在上述代码中,首先从键盘输入年龄和身高,然后判断年龄是否在 18 至 30 岁区间,并且判断身高是否在 170 至 185 区间,多个条件之间用逻辑与运算符 and 连接,也就是说,只有当所有子表达式的结果都为真,if 后面的整个表达式的结果才会为真,只要有一个子表达式的结果为假,整个表达式的结果就是假,例如当身高为 165 时,表达式 height >=170 的值为假。

3.3.3　多分支选择

多分支选择结构用于处理多个条件、多个分支的情况,可以用 if-elif-else 语句实现,其语法格式如下:

```
if 表达式 1:
   执行语句块 1
elif 表达式 2:
   执行语句块 2
elif 表达式 3:
   执行语句块 3
……
else:
  执行语句 n
```

表达式 1、表达式 2、……、表达式 n 表示条件,它们的值为布尔值,在这些表达式后面要加上冒号;语句块 1、语句块 2、……、语句块 n 可以是单个语句或多个语句,这些语句必须向右缩进,而且语句块中包含的多个语句必须具有相同的缩进量。

多分支选择结构的流程图如图 3-25 所示。

由图 3-25 可以看出,if-elif-else 语句的执行流程如下:首先计算表达式 1 的值,如果表达式 1 的值为 True,则执行语句块 1,否则计算表达式 2 的值;如果表达式 2 的值为 True,则执行语句块 2,否则计算表达式 3 的值,以此类推。如果所有表达式的值均为 False,则执行 else 后面的语句块 n。

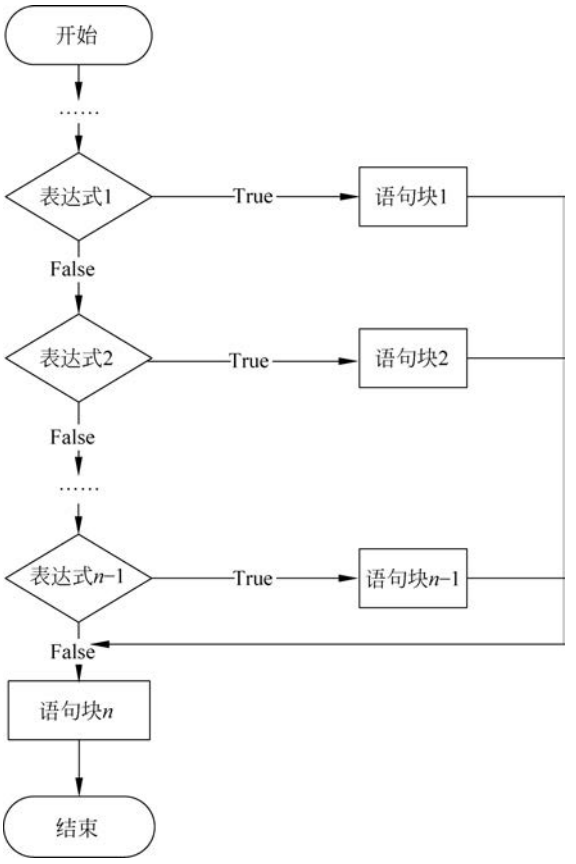

图 3-25　多分支选择结构流程图

【例 3-10】 按百分制输入学生成绩,然后将成绩划分为 5 个等级:60 分以下为不及格,60～69 分为及格,70～79 分为中等,80～89 分为良好,90 分及以上为优秀。要求从键盘输入成绩后输出相应等级。

算法分析:输入的学生成绩可使用浮点数表示并按标准划分为 5 个分数段,每个分数段对应一个等级,形成 5 个分支,可以通过多分支选择结构的 if 语句进行处理,也可以通过多个单分支选择结构的 if 语句进行处理。当使用多分支结构时,由于各个分支的条件相互排斥,所以代码更为简洁,代码如下:

```
#第 3 章/3 - 10.py
#多分支选择结构

score = float(input("请输入百分制分数: "))
if score < 60:
   grade = "不及格"
elif score < 70:      #此处是在前面条件不成立的情况下,即理解为 60 <= score < 70
   grade = "及格"
```

```
elif score < 80:      #此处是在前面条件不成立的情况下,即理解为 70 <= score < 80
   grade = "中等"
elif score < 90:      #此处是在前面条件不成立的情况下,即理解为 80 <= score < 90
   grade = "良好"
else:                 #此处是在前面条件不成立的情况下,即理解为 score >= 90
   grade = "优秀"
print("学生成绩: %.1f; 成绩等级: %s" %(score, grade))
```

上述代码的执行结果如图 3-26 所示。

```
请输入百分制分数: 85.5
学生成绩: 85.5; 成绩等级: 良好
```

图 3-26　多分支选择结构判断学生成绩等级

3.3.4　选择嵌套

当使用选择结构控制程序的执行流程时,如果有多个条件并且条件之间存在递进关系,则可以在一个选择结构中嵌入另一个选择结构,由此形成选择结构的嵌套。在内层的选择结构中还可以继续嵌入选择结构,嵌套的深度是没有限制的。

在 if-else 语句中嵌入 if-else 语句的语法格式如下:

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

在 if-else 语句中嵌入 if 语句的语法格式如下:

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

在第 1 个嵌套结构中,else 与第 2 个 if 配对;第 2 个嵌套结构中,else 与第 1 个 if 配对。也就是说,在使用嵌套的选择结构时系统将根据代码的缩进量来确定代码的层次关系。

【例 3-11】 编写一个登录程序。从键盘输入用户名和密码,然后对输入的用户名进行验证,如果用户名正确,则再对输入的密码进行验证。

算法分析:由于要求先验证用户名后验证密码,因此在程序中可以使用嵌套的选择结构,即在外层 if 语句中验证用户名,如果用户名正确无误,则再进入内层 if 语句验证密码,代码如下:

```
#第 3 章/3 - 11.py
#选择嵌套
```

```
#设置用户名和密码
USERNAME = 'admin'                          #设置用户名的值
PASSWORD = 'pwd123'                         #设置密码的值
username = input('请输入用户名:')           #从键盘输入用户名
if username == USERNAME:                    #验证用户名
    password = input('请输入密码:')         #从键盘输入密码
    if password == PASSWORD:
        print('登录成功,欢迎 %s 进入系统!' %(username))
    else:
        print('密码错误,登录失败!')
else:
    print('用户名 %s 不存在,登录失败!' %(username))
```

上述代码的执行结果如图 3-27 所示。

```
请输入用户名:administrator
用户名administrator不存在，登录失败!

请输入用户名:admin
请输入密码:123
密码错误，登录失败!
请输入用户名:admin
请输入密码:pwd123
登录成功，欢迎admin进入系统!
```

图 3-27　选择嵌套判断用户账号和密码

根据上述程序的代码缩进量,可得知第 1 个 else 与第 2 个 if 配对,第 2 个 else 与第 1 个 if 配对。在外层选择嵌套中,判断用户名是否正确,当外层选择嵌套中 if 成立,才进入内层选择嵌套判断密码是否正确。

3.4　循环结构

循环结构是控制一个语句块重复执行的程序结构,它由循环体和循环条件两部分组成,其中循环体是重复执行的语句块,循环条件则是控制是否继续执行该语句块的表达式。循环结构的特点是在一定条件下重复执行某些语句,直至重复到一定次数或该条件不再成立为止。

在 Python 语言中,可以通过 while 语句和 for 语句实现循环结构,也可以通过 break 语句、continue 语句及 pass 语句对循环结构的执行过程进行控制,此外还可以在一个循环结构中使用另一个循环结构,从而形成循环结构的嵌套。

3.4.1　while 循环

Python 编程中 while 语句用于循环执行程序,即在某条件下,循环执行某段程序,以处理需要重复处理的相同任务,其语法格式如下:

```
while 表达式:
    执行语句块……
```

执行语句块可以是单个语句或语句块。表达式可以是任何表达式，任何非零或非空(null)的值均为 True。当表达式的值为 False 时，循环结束。

表达式表示循环条件，它通常是关系表达式或逻辑表达式，也可以是结果能够转换为布尔值的任何表达式，表达式后面必须添加冒号。语句块是重复执行的单个或多个语句，称为循环体。当循环体只包含单个语句时，也可以将该语句与 while 写在同一行；当循环体包含多个语句时，这些语句必须向右缩进，而且具有相同的缩进量。

while 循环的执行流程图如图 3-28 所示。

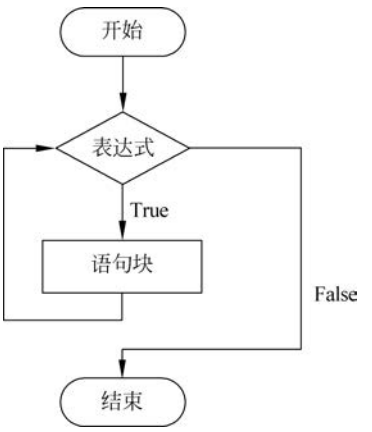

图 3-28　while 循环流程图

由图 3-28 可以看出，while 语句的执行流程如下：首先计算表达式的值，如果该值为 True，则重复执行循环体中的语句块，直至表达式的值变为 False 才结束循环，接着执行 while 语句的后续语句。

在 while 语句中，如果条件表达式的值恒为 True，则循环将无限次地执行下去，这种情况称为死循环。为了避免出现死循环，必须在循环体内包含能修改条件表达式值的语句，使该值在某个时刻变为 False，从而结束循环。

在 Python 中，允许在循环语句中使用可选的 else 子句，即

```
while 表达式:
    执行语句块 1……
else:
    执行语句块 2……
```

其中，语句块 2 可以包含单个或多个语句，这些语句将在循环正常结束的情况下执行。如果通过 break 语句中断循环，则不会执行语句块 2 中的语句。

【例 3-12】 计算整数 1 到 100 的和，用 while 语句实现，代码如下：

```
#第 3 章/3 - 12.py
#while 循环

i = 1          #初始化一个变量
m = 0
while i <= 100:
    m += i
    i += 1
print("1 到 100 的和: ",m)
```

上述代码的执行结果如图 3-29 所示。

```
1到100的和:  5050
```

图 3-29　while 循环计算 1 到 100 的和

将变量 i 的初始值设置为 1，m 的初始值为 0，循环条件为 i <= 100，在循环体中将每个数累加起来并存入变量 m 中，并使变量 i 增加 1。待循环正常结束后，用 print()函数输出 m 的值即可。

3.4.2 for 循环

for 循环是 Python 中的另外一种循环语句，提供了 Python 中最强大的循环结构，它可以循环遍历多种序列项目，如一个列表或者一个字符串，其语法格式如下。

```
for 循环变量 in 序列对象:
    语句块
```

其中，循环变量不需要事先进行初始化。序列对象用于指定要遍历的字符串、列表、元组、集合或字典。语句块表示循环体，可以包含单个或多个语句。当循环体只包含单个语句时，也可以将这个语句与 for 写在同一行；当循环体包含多个语句时，这些语句必须向右缩进，而且必须具有相同的缩进量。

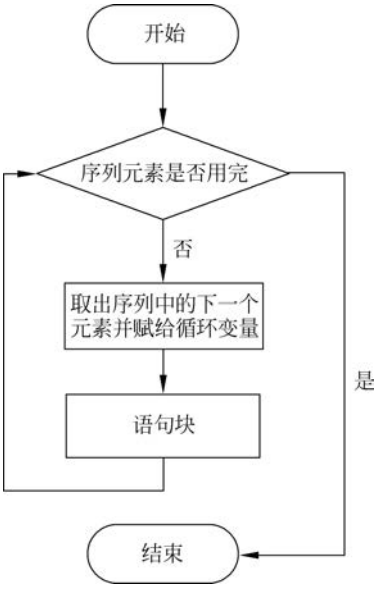

图 3-30 for 循环流程图

for 循环的执行流程图如图 3-30 所示。

for 语句的执行流程如下：将序列对象中的元素依次赋给循环变量，并针对当前元素执行一次循环体语句块，直至序列中的每个元素都已用过，遍历结束为止。

与 while 语句一样，在 for 语句中也可以使用一个可选的 else 子句。当 for 循环正常结束时将会执行 else 子句。如果通过执行 break 子句而中断 for 循环，则不会执行 else 子句。

【例 3-13】 从键盘输入一个自然数，判断该数是否是素数，用 for 语句实现。

算法分析：素数是指一个数只能被 1 和自身整除，再没有其他因子。如果一个数被比它小的数(1 除外)整除后余数为 0，则说明这个数可以被除了 1 和自身的其他数整除，则该数是合数。

```
#第 3 章/3 - 13.py
#for 循环

x = int(input('请输入一个数:'))
m = x//2
for i in range(2,m):
    if x % i == 0:
        print('%d 是合数'%(x))
        break
else:
    print('%d 是素数'%(x))
```

上述代码的执行结果如图 3-31 所示。

```
请输入一个数:17
17是素数
```

```
请输入一个数:81
81是合数
```

图 3-31　for 循环判断素数

在上述代码中，从键盘输入字符型数字转换为整数后，把该整数整除了 2。因为如果一个数是合数，则一定能够在 2 到这个整数的一半之间找到一个因子，因此循环时，in 后面的序列只需从 2 开始到这个整数的一半结束，让该整数去除以这个序列中的每个数，然后求余数，只要有一个余数是 0，则 if 语句成立，打印该数是合数，并执行 break 语句，结束整个 for 循环，并且不会执行 break 之后的 else 语句。如果 for 循环结束都没有找到因子，则执行 else 语句。else 语句与 for 循环为同等缩进，意味着是同一级别，在 for 循环执行后执行。

3.4.3　嵌套循环

Python 语言支持嵌套循环，所谓嵌套循环就是一个外循环的主体部分是一个内循环。内循环或外循环可以是任何类型，例如 while 循环或 for 循环。外部 for 循环可以包含一个 while 循环，反之亦然。外循环可以包含多个内循环。

在嵌套循环中，迭代次数将等于外循环中的迭代次数乘以内循环中的迭代次数。在外循环的每次迭代中，内循环执行其所有迭代。对于外循环的每次迭代，内循环重新开始并在外循环可以继续下一次迭代之前完成其执行。

1. for 嵌套循环

Python for 嵌套循环的语法格式如下：

```
for 循环变量 in 序列对象:
    for 循环变量 in 序列对象:
        语句块
```

【例 3-14】 利用嵌套循环打印 100 以内的所有素数，代码如下：

```
#第 3 章/3 - 14.py
#Python for 嵌套循环

num = []          #定义一个空列表,用来接收找到的符合条件的数字
for i in range(2, 101):
    k = 0
    for j in range(2, i + 1):
        if i % j == 0:
            k += 1
    if k == 1:
        num.append(i)
print('100 以内的所有素数: ',num)
```

上述代码的执行结果如图 3-32 所示。

```
100以内的所有素数:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

图 3-32　for 嵌套循环找出 100 以内的所有素数

在上述代码中,range(2,101)返回一个 2 到 100 的整数序列,所以外部 for 循环是从 2 到 100 迭代数字。在嵌套循环的第 1 次迭代中,数字是 2。下一次,它是 3。以此类推,直到 100。接下来,对于外循环的每次迭代,例如当外循环的循环变量 i 等于 10 时,内循环变量 j 将从 2 执行到 10,当外循环的循环变量 i 等于 20 时,内循环变量 j 将从 2 执行到 20。

2. while 循环嵌套

Python while 循环嵌套的语法格式如下:

```
while 表达式:
    while 表达式:
        执行语句块……
```

【例 3-15】 利用嵌套循环打印九九乘法表。

算法分析:输出乘法口诀表可以通过一个二重循环实现,外层循环需要执行 9 次,每执行一次输出一行;各外层循环输出的结果位于不同的行。内层循环执行的次数由行号决定,行号是多少内层循环就执行多少次,每执行一次输出一个等式,同一个内层循环输出的所有等式位于同一行,代码如下:

```
#第 3 章/3 - 15.py
#Python while 嵌套循环

i = 1
while i <= 9:
    a = 1
    while a <= i:
        print("%d * %d = %d" %(i, a, i * a), end = '')
        a += 1
    print('')
    i += 1
```

上述代码的执行结果如图 3-33 所示。

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

图 3-33 while 嵌套循环打印九九乘法表

在上述代码中,外部 while 循环是从 1 到 9 迭代数字,所以外循环的执行次数是 9,代表了九九乘法表的九行。在嵌套循环的第 1 次迭代中,数字是 1。下一次,它是 2。以此类推,直到 9。接下来,对于外循环的每次迭代,例如当外循环的循环变量 i 等于 5 时,内循环变量 a 将从 1 执行到 5,当外循环的循环变量 i 等于 8 时,内循环变量 a 将从 1 执行到 8,代表了九九乘法表的当前行将要打印的列数。在内部循环的每次迭代中,计算两个数字的乘法。

以上两个程序代码，也可以在循环体内嵌入其他的循环体，如在 while 循环中可以嵌入 for 循环，反之，可以在 for 循环中嵌入 while 循环。

3.4.4　循环控制

循环语句的正常执行流程是在满足循环条件时执行循环体，一旦循环条件不再满足便会执行 else 子句或者继续执行循环语句的后续语句。如果需要改变循环的执行流程，则可以使用 Python 提供的以下 3 个循环控制语句。各控制语句的描述见表 3-9。

表 3-9　Python 控制语句

控 制 语 句	说　　明
break 语句	在语句块的执行过程中终止循环，并且跳出整个循环
continue 语句	在语句块的执行过程中终止当前循环，跳出该次循环，继续执行下一次循环
pass 语句	pass 是空语句，其作用是保持程序结构的完整性

1. break 语句

break 语句用来终止当前循环的执行操作，其语法格式如下：

```
break
```

break 语句用在 while 和 for 循环中，通常与 if 语句一起使用，可以用来跳出当前所在的循环结构，即使循环条件表达式的值没有变成 False 或者序列还没被完全遍历完，系统也会立即停止执行循环语句，即跳出循环体，如果存在 else 子句，则将跨过 else 子句，转而执行循环语句的后续语句。break 语句的流程图如图 3-34 所示。

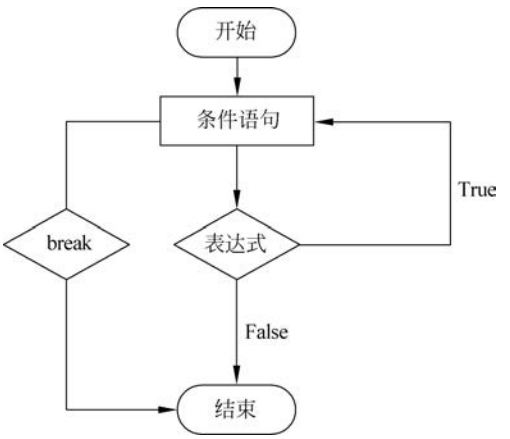

图 3-34　break 语句流程图

以下代码演示了 Python break 语句中止 for 循环的操作：

```
#使用 break 语句中止 for 循环
```

```
for letter in 'Python':
   if letter == 'h':
     break
   print('当前字母:', letter)
```

上述代码的执行结果如图 3-35 所示。

在上述代码中,当循环变量 letter 的值不等于字母 h 时,打印出当前循环变量的值,当循环变量 letter 的值等于字母 h 时,break 退出 for 循环。

break 语句中止 while 循环的操作代码如下:

```
#使用 break 语句中止 while 循环

var = 10
while var > 0:
  print('当前变量的值:', var)
  var = var - 1
  if var == 6: #当变量 var 等于 6 时退出循环
    break
```

上述代码的执行结果如图 3-36 所示。

```
当前字母: P
当前字母: y
当前字母: t
```

图 3-35 break 终止 for 循环

```
当前变量的值 : 10
当前变量的值 : 9
当前变量的值 : 8
当前变量的值 : 7
```

图 3-36 break 终止 while 循环

在上述代码中,每次进入循环体,首先打印出当前变量的值,再修改循环变量的值,当循环变量 var 的值等于 6 时,break 退出 while 循环。

2. continue 语句

continue 语句用于跳出本次循环,其语法格式如下:

```
continue
```

与 break 语句一样,continue 语句也用在 while 和 for 循环中,通常与 if 语句一起使用,但两者的作用有所不同。continue 语句用来跳过当前循环的剩余语句,然后继续进行下一轮循环;break 语句则是跳出整个循环,然后继续执行循环语句的后续语句。continue 语句的流程图如图 3-37 所示。

continue 语句中止 for 循环的操作代码如下:

```
#使用 continue 语句中止 for 循环

for letter in 'Python':
   if letter == 'h':
     continue
   print('当前字母:', letter)
```

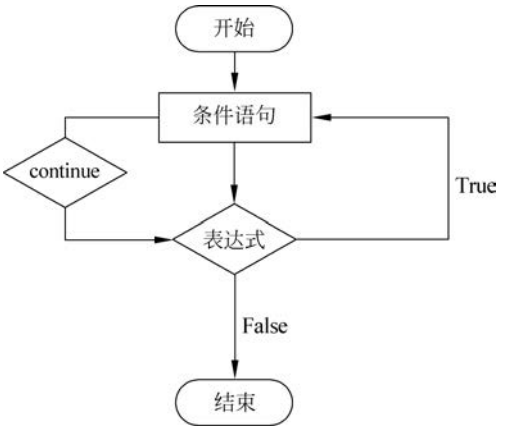

图 3-37　continue 语句流程图

上述代码的执行结果如图 3-38 所示。

在上述代码中，当循环变量 letter 的值不等于字母 h 时，打印出当前循环变量的值，当循环变量 letter 的值等于字母 h 时，continue 退出当前循环，继续下一次循环。

continue 语句中 while 终止循环的操作代码如下：

```
var = 10
while var > 0:
   var = var - 1
   if var == 6:
     continue
   print '当前变量的值:', var
```

上述代码的执行结果如图 3-39 所示。

```
当前字母: P
当前字母: y
当前字母: t
当前字母: o
当前字母: n
```

图 3-38　continue 循环终止 for 循环

```
当前变量的值: 9
当前变量的值: 8
当前变量的值: 7
当前变量的值: 5
当前变量的值: 4
当前变量的值: 3
当前变量的值: 2
当前变量的值: 1
当前变量的值: 0
```

图 3-39　continue 循环终止 for 循环

在上述代码中，每次进入循环体，首先修改循环变量的值，再打印出当前变量的值，当循环变量 var 的值等于 6 时，continue 退出当前循环，继续下一次循环。

3. pass 语句

为了保持程序结构的完整性，Python 提供了一个空语句 pass。pass 语句一般作为占位语句使用，不做任何其他事情，其语法格式如下：

```
pass
```

以下代码演示了 pass 语句占位的操作：

```
#使用 pass 语句占位

for letter in 'Python':
   if letter == 'h':
     pass
     print('这是 pass 块')
   print('当前字母:', letter)
```

```
当前字母: P
当前字母: y
当前字母: t
这是 pass 块
当前字母: h
当前字母: o
当前字母: n
```

图 3-40　pass 占位

上述代码的执行结果如图 3-40 所示。

在上述代码中，当循环变量 letter 的值等于字母 h 时，遇到 pass 语句，其作用是占用一行语句位置。

【例 3-16】 输入员工的薪资，若薪资小于 0，则重新输入。最后打印出录入员工的数量和薪资明细，以及平均薪资。

代码如下：

```
#第 3 章/3-16.py
#循环控制

salarySum = 0
salarys = []
for i in range(4):
   s = input("请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):")
   if s.upper() == 'Q':
      print("录入结束,退出")
      break
   if float(s) < 0:
      continue
   salarys.append(float(s))
   salarySum += float(s)
else:
   print("您已经全部录入 4 名员工的薪资")
print("录入薪资: ", salarys)
print("平均薪资 %1.f" %(salarySum/4))
```

上述代码的执行结果如图 3-41 所示。

```
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):5600
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):5200
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):6800
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):q
录入结束，退出
录入薪资: [5600.0, 5200.0, 6800.0]
您录入了3名员工工资，平均薪资5867
```

```
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):5600
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):5200
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):6800
请输入一共 4 名员工的薪资(按 Q 或 q 中途结束):6000
您已经全部录入 4 名员工的薪资
录入薪资: [5600.0, 5200.0, 6800.0, 6000.0]
您录入了4名员工工资，平均薪资5900
```

图 3-41　循环控制计算员工平均工资

在上述代码中，range(4)返回 0～3 的序列，循环变量 i 的值从 0 循环至 3，当输入的内容等于大写字母 Q 或者小写字母 q 时，break 会退出循环并跳过 else 语句，如图 3-41 所示；若循环变量 i 遍历完序列的每个值，则会执行 else 语句。使用 len()方法可以直接得到列表元素的个数。

注意：对于带有 else 子句的循环语句，如果是因为循环条件表达式不成立而自然结束循环，则执行 else 子句中的代码。

3.5　综合案例：阶梯电价计算电费

为了提倡居民节约用电，某省电力公司执行“阶梯电价”。该省的基础电价为 0.56 元/kWh，居民阶梯电价总电费＝第一档电费＋第二档电费＋第三档电费。

夏季标准(5～10 月)：

第一档电量：每户每月 200kWh，电价不作调整；

第二档电量：每户每月 200kWh 以上，500kWh 以内，电价每千瓦时加价 0.05 元；

第三档电量：每户每月 500kWh 以上，电价每千瓦时加价 0.30 元。

非夏季标准(1～4 月、11～12 月)：

第一档电量：每户每月 200kWh，电价不作调整；

第二档电量：每户每月 200kWh 以上，400kWh 以内，电价每千瓦时加价 0.08 元；

第三档电量：每户每月 400kWh 以上，电价每千瓦时加价 0.50 元。

【要求】

输入当前月份和当月用电总量，计算该月实际应缴纳的电费。

【目标】

通过该案例的训练，熟悉 Python 运算符、流程控制的用法，掌握 Python 的多分支选择结构、选择嵌套、循环结构、循环控制等常用流程控制的基本操作和应用场景，培养实际动手能力和解决问题的能力。

【步骤】

(1) 让用户输入一个月份数字，如果该数字不在 1～12 月，则循环让用户重新输入，直到输入了有效的月份，使用 break 退出循环，代码如下：

```
#输入有效的月份
while(True):
    month = int(input("请选择当前的月份(1～12): "))
    if month >= 1 and month <= 12:
        break
    else:
        continue
```

(2) 让用户输入当月用电总量,如果该数字小于0,则循环让用户重新输入,直到输入了有效的用电量,使用 break 退出循环,代码如下:

```
#输入有效的用电量
while(True):
    sumDegrees = float(input("请输入您当月所使用的用电量: "))
    if sumDegrees >= 0:
        break
    else:
        continue
```

(3) 判断用户输入的月份是夏季还是非夏季,作为嵌套选择结构中外层选择结构的 if 表达式。in 后面的表达式是一个列表对象,代码如下:

```
#输入有效的度数
if month in[1,2,3,4,11,12]:
  ……
else:
  ……
```

(4) 如果当前月份是非夏季,则 if month in[1,2,3,4,11,12]为真,进入 if 表达式下的内层选择结构,内层是一个多分支选择结构,判断用电总量所在的区间,从而计算出每一档的用电量,最后求出总电费。总电费 eleCharge 定义在程序的最开始位置,是一个全局变量,程序内部共同使用一个总电费变量,代码如下:

```
if sumDegrees <= 200:
        eleCharge = unitPrice * sumDegrees
elif sumDegrees > 200 and sumDegrees <= 400 :
        d1 = 200 * unitPrice
        d2 = (sumDegrees - 200) * (unitPrice + 0.08)
        eleCharge = d1 + d2
else:
        d1 = 200 * unitPrice
        d2 = 200 * (unitPrice + 0.08)
        d3 = (sumDegrees - 400) * (unitPrice + 0.5)
        eleCharge = d1 + d2 + d3
```

(5) 如果当前月份是夏季,则 else 为真,进入 else 下面的内层选择结构,内层也是一个多分支选择结构,同样需要判断用电总度数所在的区间,从而计算出每一档的用电量,最后求出总电费,代码如下:

```
if sumDegrees <= 200:
        eleCharge = unitPrice * sumDegrees
elif sumDegrees > 200 and sumDegrees <= 500:
        d1 = 200 * unitPrice
        d2 = (sumDegrees - 200) * (unitPrice + 0.05)
        eleCharge = d1 + d2
else:
        d1 = 200 * unitPrice
        d2 = 300 * (unitPrice + 0.05)
        d3 = (sumDegrees - 500) * (unitPrice + 0.3)
        eleCharge = d1 + d2 + d3
```

（6）编写程序，代码如下：

```
#第 3 章/calculateEleccharge.py
#阶梯电价计算电费

unitPrice = 0.56                    #基础电价
eleCharge = 0                       #每月应缴电费

#输入有效的月份
while(True):
    month = int(input("请选择当前的月份(1～12): "))
    if month >= 1 and month <= 12:
        break
    else:
        continue

#输入有效的用电量
while(True):
    sumDegrees = float(input("请输入您当月所使用的用电量: "))
    if sumDegrees >= 0:
        break
    else:
        continue

if month in[1,2,3,4,11,12]:     #如果当前是非夏季
    if sumDegrees <= 200:
        eleCharge = unitPrice * sumDegrees
    elif sumDegrees > 200 and sumDegrees <= 400:
```

```
            d1 = 200 * unitPrice
            d2 = (sumDegrees - 200) * (unitPrice + 0.08)
            eleCharge = d1 + d2
        else:
            d1 = 200 * unitPrice
            d2 = 200 * (unitPrice + 0.08)
            d3 = (sumDegrees - 400) * (unitPrice + 0.5)
            eleCharge = d1 + d2 + d3
    else:                                  #如果当前是夏季
        if sumDegrees <= 200:
            eleCharge = unitPrice * sumDegrees
        elif sumDegrees > 200 and sumDegrees <= 500:
            d1 = 200 * unitPrice
            d2 = (sumDegrees - 200) * (unitPrice + 0.05)
            eleCharge = d1 + d2
        else:
            d1 = 200 * unitPrice
            d2 = 300 * (unitPrice + 0.05)
            d3 = (sumDegrees - 500) * (unitPrice + 0.3)
            eleCharge = d1 + d2 + d3
    print("您%d月所使用电量的电费为%.2lf元"%(month,eleCharge))
```

上述代码的执行结果如图 3-42 所示。

```
请选择当前的月份(1~12): 4
请输入您当月所使用的用电量: 385
您4月所使用电量的电费为230.40元

请选择当前的月份(1~12): 8
请输入您当月所使用的用电量: 520.5
您8月所使用电量的电费为312.63元
```

图 3-42　流程控制实现阶梯电价计算电费

3.6　小结

本章首先介绍了 Python 语言中的各个运算符,然后对 Python 流程控制中的 3 种结构(顺序结构、选择结构和循环结构)一一做了详细讲解和举例说明,最后以一个综合案例(阶梯电价计算电费)来对本章的知识进行实例训练。

本章的知识结构如图 3-43 所示。

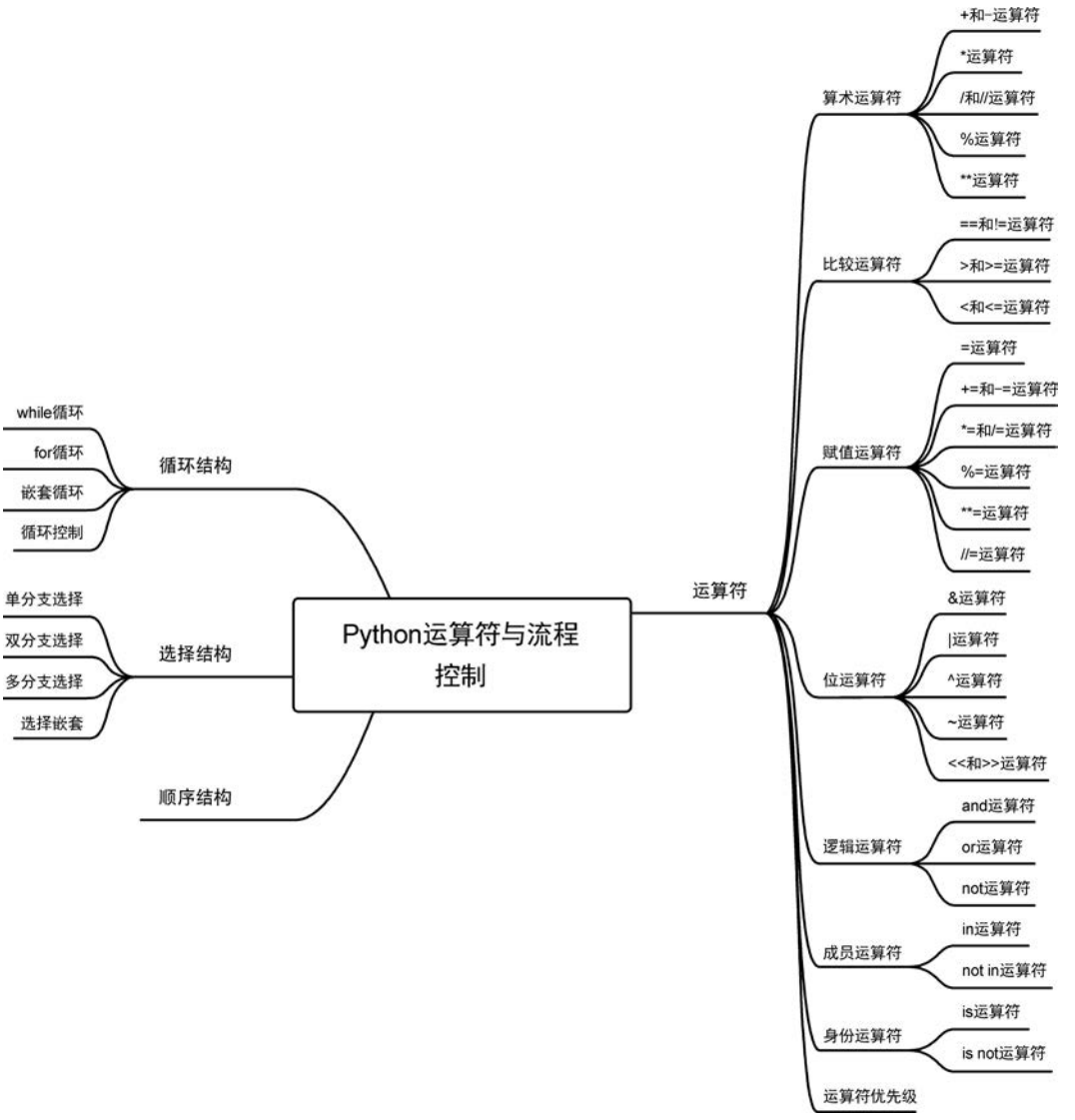

图 3-43　Python 运算符与流程控制知识结构图

3.7　习题

1. 选择题

（1）幂运算的运算符为(　　)。

A. *　　B. **　　C. %　　D. //

（2）优先级最高的运算符为(　　)。

A. /　　B. //　　C. *　　D. ()

(3) 关于 a or b 的描述错误的是(　　)。

A. 若 a=True,b=True,则 a or b==True

B. 若 a=True,b=False,则 a or b==True

C. 若 a=True,b=True,则 a or b==False

D. 若 a=False,b=False,则 a or b==False

(4) 与 x>y and y>z 语句等价的是(　　)。

A. x>y>z　　B. not x<y or not y<z

C. not x<y or y<z　　D. x>y or not y<z

(5) 下列运算符的使用错误的是(　　)。

A. 1 + 'a'　　B. [1, 2, 3] + [4, 5, 6]

C. 3 * 'abc'　　D. −10 % −3

(6) 以下可以终结一个循环的执行的语句是(　　)。

A. break　　B. if　　C. input　　D. exit

(7) 以下的布尔代数运算错误的是(　　)。

A. (True or x)==True

B. not(a and b)==not(a) and not(b)

C. (False and x)==False

D. (True or False)==True

(8) for i in range(0,2):

```
print(i)
```

上述程序的输出结果是(　　)。

A. 0 1 2　　B. 1 2　　C. 0 1　　D. 1

(9) 以下关于循环结构的描述,错误的是(　　)。

A. 遍历循环使用 for <循环变量> in 循环结构语句,其中循环结构不能是文件

B. 使用 range0 函数可以指定 for 循环的次数

C. for i in range(5)表示循环 5 次,i 的值是从 0 到 4

D. 用字符串做循环结构时,循环的次数是字符串的长度

(10) 以下关于循环控制语句描述错误的是哪一项(　　)。

A. Python 中的 for 语句可以在任意序列上进行迭代访问,例如列表、字符串和元组

B. 在 Python 中 if-elif-elif 结构中必须包含 else 子句

C. 在 Python 中没有 switch-case 关键词,可以用 if-elif-elif 来等价表达

D. 循环可以嵌套使用,例如一个 for 语句中有另一个 for 语句,一个 while 语句中有一个 for 语句等

(11) 下面代码执行后，x 的结果是(　　)。

```
x = 2
x * = 3 + 5 * 2
```

A. 13　　B. 15　　C. 26　　D. 8192

(12) print(100 − 25 * 3 % 4)应该输出(　　)。

A. 1　　B. 97　　C. 25　　D. 0

(13) 下列表达式的值为 True 的是(　　)。

A. 3 > 2 > 2　　B. 1 and 2 != 1

C. not(11 and 0 != 2)　　D. 10 < 20 and 10 < 5

(14) 设 x=10、y=20，下列语句能正确运行结束的是(　　)。

A. max = x > y ? x : y　　B. if(x > y) print(x)

C. while True: pass　　D. min = x if x < y else y

(15) 以下关于 Python 的控制结构，错误的是(　　)。

A. 每个 if 条件后要使用冒号(:)

B. 在 Python 中，没有 switch-case 语句

C. Python 中的 pass 是空语句，一般用作占位语句

D. elif 可以单独使用

(16) 关于 Python 循环结构，以下选项中描述错误的是(　　)。

A. Python 通过 for、while 等保留字构建循环结构

B. 遍历循环中的遍历结构可以是字符串、文件、组合数据类型和 range0 函数等

C. break 用来结束当前当次语句，但不跳出当前的循环体

D. continue 只结束本次循环

(17) 如果用户输入的整数不合规将会导致程序出错，为了不让程序异常中断，则需要用到的语句是(　　)。

A. if 语句　　B. eval 语句

C. 循环语句　　D. try-except 语句

(18) 下面 Python 循环体执行的次数与其他不同的是(　　)。

A.
```
i = 0
while( i <= 10):
    print(i)
    i = i + 1
```

B.
```
i = 10
while(i > 0):
    print(i)
    i = i - 1
```

C.
```
for i in range(10):
    print(i)
```

D.
```
for i in range(10, 0, -1):
    print()
```

(19) 执行下列 Python 语句将产生的结果是(　　)。

```
x = 2; y = 2.0
if(x = = y): print("Equal")
else: print("Not Equal")
```

A. Equal　　B. Not Equal　　C. 编译错误　　D. 运行时错误

(20) 下面 if 语句统计满足“性别(gender)为男、职称(rank)为教授、年(age)小于 40 岁”条件的人数,正确的语句为(　　)。

A. if(gender ==="男" or age<40 and rank ==="教授"): n+=1

B. if(gender ==="男" and age<40 and rank==="教授"): n+=1

C. if(gender ==="男" and age<40 or rank==="教授"): n+=1

D. if(gender ==="男" or age<40 or rank==="教授"): n+=1

2. 判断题

(1) 已知 x=3,那么执行语句 x+=6 之后,x 的内存地址不变。　　(　　)

(2) 如果仅用于控制循环次数,则使用 for i in range(20)和 for i in range(20,40)的作用是等价的。　　(　　)

(3) 在编写多层循环时,为了提高运行效率,应尽量减少内循环中不必要的计算。(　　)

(4) Python 运算符%不仅可以用来求余数,还可以用来格式化字符串。　　(　　)

(5) 表达式 pow(3,2)==3 ** 2 的值为 True。　　(　　)

(6) 在 Python 中可以使用 if 作为变量名。　　(　　)

(7) 表达式 5 if 5>6 else(6 if 3>2 else 5)的值为 5。　　(　　)

(8) while 循环如果设计不小心,则会出现死循环。　　(　　)

(9) for 或者 while 与 else 搭配使用时,循环正常结束后执行。　　(　　)

(10) 单分支结构是用 if 保留字判断满足一个条件,就执行相应的处理代码,双分支结构是用 if-else 根据条件的真假,执行两种处理代码,多分支结构是用 if-elif-else 处理多种可能的情况。　　(　　)

3. 写出下列程序的运行结果

(1)

```
for num in range(2,10):
      if num % 2 == 0:
          continue
      print("Find an odd number", num)
```

(2)

```
for i in "the number changes":
   if i == 'n':
      break
   else:
      print(i, end = " ")
```

(3)

```
for i in range(10):
   if i % 2 == 0:
```

```
        continue
    else:
        print(i, end = " ")
```

(4)

```
sum = 1.0
for num in range(1,4):
    sum += num
print(sum)
```

(5)

```
for i in range(3):
    for s in "abcd":
        if s == "c":
            break
        print(s, end = "")
```

(6)

```
age = 15
start = 2
if age % 2 != 0:
     start = 1
for x in range(start, age + 2, 2):
     print(x)
```

(7)

```
for n in range(100,500):
    i = n//100
    j = n//10 % 10
    k = n % 10
    if n == i ** 3 + j ** 3 + k ** 3:
       print(n)
```

(8)

```
a = 30
b = 1
if a >= 10:
    a = 20
elif a >= 20:
    a = 30
elif a >= 30:
    b = a
```

```
else:
    b = 0
print('a = {}, b = {}'.format(a,b))
```

(9)

```
for i in"CHINA":
    for k in range(2):
        print(i, end = "")
        if i == 'N':
            break
```

(10)

```
count = 1
tag = True
while tag:
    if count < 20:
        count += 1
        print(count)
        if count % 2 == 0:
            count += 1
            continue
```

4. 编程题

(1) 编写程序,利用循环结构求 n!。

(2) 打印 0～100 的奇数,利用 continue 语句完成。

(3) 编写程序,使用嵌套的 if 语句,输出 2000～3000 的所有闰年。

(4) 如果列出 10 以内自然数中 3 或 5 的倍数,则包括 3、5、6、9。那么这些数字的和为 23。要求计算得出任意正整数 n 以内中 3 或 5 的倍数的自然数之和。

(5) 如果某自然数除它本身之外的所有因子之和等于该数,则该数被称为完数。输出 1000 以内的所有完数。

(6) 10 以内的素数 2、3、5、7 的和为 17。要求计算得出任意正整数 n 以内的所有素数的和。

(7) 数字 197 可以被称为循环素数,因为 197 的 3 个数位循环移位后的数字 197、971、719 均为素数。100 以内这样的数字共有 13 个,如 2、3、5、7、11、13、17、31、37、71、73、79、97。求出任意正整数 n 以内一共有多少个这样的循环素数。

(8) 编写程序,显示 100～200 不能被 3 整除的数。每行显示 10 个数。程序的运行结果如下所示。

100 101 103 104 106 107 109 110 112 113

115 116 118 119 121 122 124 125 127 128

130 137 131 133 134 136 139 140 142 143
145 157 146 148 149 151 152 154 155 158
160 161 163 164 166 167 169 170 172 173
175 176 178 179 181 182 184 185 187 188
190 191 193 194 196 197 199 200

(9) 编写程序,实现分段函数的计算,如表所示。

x	y
x＜0	0
0＜=x＜5	x
5＜=x＜10	3x－5
10＜=x＜20	x－2
20＜=x	0

(10) 棋盘放麦粒,在印度有一个古老的传说,舍罕王打算奖赏国际象棋的发明人——宰相西萨·班·达依尔。国王问宰相想要什么,他对国王说:“陛下,请您在这张棋盘的第1个小格里,赏给我1粒麦子,在第2个小格里给2粒,第3小格里给4粒,以后每小格都比前一小格加一倍的麦子数量。请您把这样摆满棋盘上所有64格的麦粒都赏给您的仆人吧!”国王觉得这要求太容易满足了,就命令将这些麦粒赏给宰相。当人们把一袋一袋的麦子搬来开始计数时,国王才发现,就是把全印度甚至全世界的麦粒全拿来,也满足不了宰相的请求。编写程序显示宰相所要求得到的麦粒总数。