第4章 程序控制结构

第4章 程序控制结构

- 有了合适的数据类型和数据结构之后，还要依赖于**选择**和**循环结构**来实现特定的业务逻辑。
- 一个完整的选择结构或循环结构可以看作是一个大的“语句”，从这个角度来讲，程序中的多条“语句”是顺序执行的。

4.1 条件表达式

- 在选择和循环结构中，都要根据**条件表达式**的值来确定下一步的执行流程。
- 条件表达式的值只要不是False、0（或0.0、0j等）、空值None、空列表、空元组、空集合、空字典、空字符串、空range对象或其他空迭代对象，Python解释器均认为与True等价。

```
>>> if 3>2:
        print("Yes")


Yes
```

4.1 条件表达式

（1）关系运算符

Python中的关系运算符可以连续使用，这样不仅可以减少代码量，也比较符合人类的思维方式。

```
>>> print(1<2<3)                    #等价于1<2 and 2<3
True
>>> print(1<2>3)
False
>>> print(1<3>2)
True
```

4.1 条件表达式

- 在Python语法中，条件表达式中不允许使用赋值运算符“=”，避免了误将关系运算符“==”写作赋值运算符“=”带来的麻烦。在条件表达式中使用赋值运算符“=”将抛出异常，提示语法错误。

```
>>> if a=3:                                    #条件表达式中不允许使用赋值运算符
SyntaxError: invalid syntax
>>> if (a=3) and (b=4):
SyntaxError: invalid syntax
```

4.1 条件表达式

（2）逻辑运算符

逻辑运算符and和or具有短路求值或惰性求值的特点，可能不会对所有表达式进行求值，而是只计算必须计算的表达式的值，可以大幅度提高程序运行效率。

```
>>> if a>5 or 3:
        print("Yes")


Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    if a>5 or 3:
NameError: name 'a' is not defined
```

```
>>> if 3 or a>5:
        print("Yes")

Yes

>>> if False and b<7:
        print("OK")
else:
        print("No")


No
```

4.2 选择结构

- 常见的选择结构有：
  - 单分支选择结构
  - 双分支选择结构
  - 多分支选择结构
  - 嵌套的分支结构
- 循环结构和异常处理结构中也可以带有“else”子句，可以看作是特殊形式的选择结构。

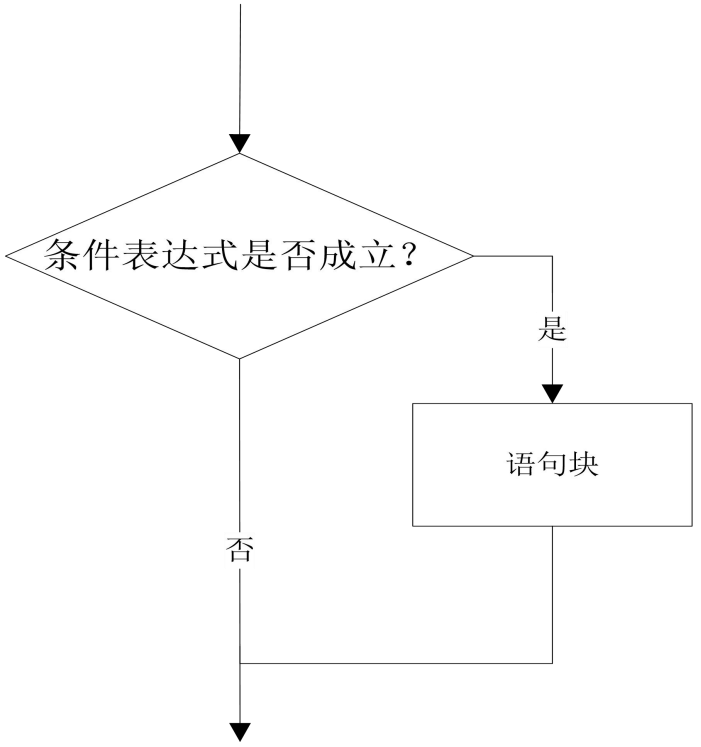
4.2.1 单分支选择结构

单分支选择结构语法：

```
if 表达式:
    语句块
```

冒号“:”是不可缺少的，表示一个语句块的开始
语句块必须做相应的缩进，一般是以4个空格为缩进单位

```
x = input('Input two number:')
a, b = map(int, x.split())
if a > b:
    a, b = b, a     #序列解包，交换两个变量的值
print(a, b)
```

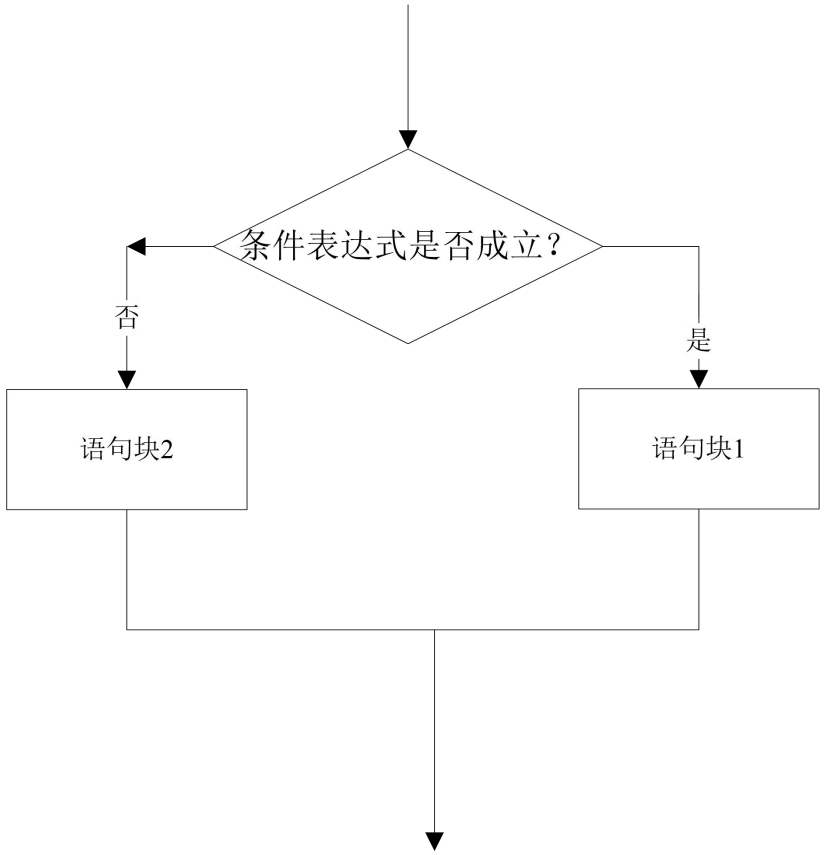


在Python中，代码的缩进非常重要，缩进是体现代码逻辑关系的重要方式，同一个代码块必须保证相同的缩进量。

4.2.2 双分支选择结构

双分支选择结构语法：

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




```
>>> chTest = ['1', '2', '3', '4', '5']
>>> if chTest:
    print(chTest)
else:
    print('Empty')

['1', '2', '3', '4', '5']
```

4.2.2 双分支选择结构

- 问题解决：鸡兔同笼(输入鸡兔总数和腿总数，求鸡和兔的数量)。
- 提示：tu = (tui - jitu*2) / 2

```
jitu, tui = map(int, input('请输入鸡兔总数和腿总数：').split())
#jitu, tui = eval(input('请输入鸡兔总数和腿总数：'))
tu = (tui - jitu*2) / 2
if int(tu) == tu:
    print('鸡：{0},兔：{1}'.format(int(jitu-tu), int(tu)))
else:
    print('数据不正确，无解')

#split参考P157
#format参考P154
```

4.2.2 双分支选择结构

- Python还提供了一个三元运算符，并且在三元运算符构成的表达式中还可以嵌套三元运算符，可以实现与选择结构相似的效果。语法为

  value1 if condition else value2

- 当条件表达式condition的值与True等价时，表达式的值为value1，否则表达式的值为value2。

```
>>> b = 6 if 5>13 else 9          #赋值运算符优先级非常低
>>> b
9
```

4.2.3 多分支选择结构

多分支选择结构的语法为：

```
if 表达式1:
    语句块1
elif 表达式2:
    语句块2
elif 表达式3:
    语句块3
……
else:
    语句块4
```

其中，关键字elif是else if的缩写。

4.2.3 多分支选择结构

- 问题解决：使用多分支选择结构将成绩从百分制变换到等级制。

```
def func(score):
    if score > 100 or score < 0:
        return 'wrong score.must between 0 and 100.'
    elif score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    elif score >= 70:
        return 'C'
    elif score >= 60:
        return 'D'
    else:
        return 'E'
```

4.2.4 选择结构的嵌套

选择结构的嵌套，格式如下：

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

注意：缩进必须要正确并且一致。

4.2.4 选择结构的嵌套

- 问题解决：使用嵌套选择结构将成绩从百分制变换到等级制。

```
def func(score):
    degree = 'DCBAAE'
    if score > 100 or score < 0:
        return 'wrong score.must between 0 and 100.'
    else:
        index = (score - 60) // 10
        if index >= 0:
            return degree[index]
        else:
            return degree[-1]
```

4.3 循环结构

- Python主要有**for循环**和**while循环**两种形式的循环结构，多个循环可以嵌套使用，并且还经常和选择结构嵌套使用来实现复杂的业务逻辑。
  - while循环一般用于循环次数难以提前确定的情况，当然也可以用于循环次数确定的情况；
  - for循环一般用于循环次数可以提前确定的情况，尤其适用于枚举或遍历序列或迭代对象中元素的场合。

4.3 循环结构

- 对于**带有else子句的循环结构**，如果循环因为条件表达式不成立或序列遍历结束而自然结束时则执行else结构中的语句，如果循环是因为执行了break语句而导致循环提前结束则不会执行else中的语句。

4.3.1 for循环与while循环

- 两种循环结构的完整语法形式分别为：

```
while 条件表达式：
    循环体
[else:
    else子句代码块]
```

和

```
for 取值 in 序列或迭代对象：
    循环体
[else:
    else子句代码块]
```

4.3.1 for循环与while循环

- 问题解决：使用循环结构遍历并输出列表中的所有元素。

```
a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
for i, v in enumerate(a_list):
    print('列表的第', i+1, '个元素是：', v)
```

4.3.1 for循环与while循环

- 问题解决：输出1~100之间能被7整除但不能同时被5整除的所有整数。

```
for i in range(1, 101):
    if i%7==0 and i%5!=0:
        print(i)
```

4.3.1 for循环与while循环

- 问题解决：计算1+2+3+…+99+100的结果。

```
s = 0
for i in range(1, 101):                    #不包括101
    s += i
else:
    print(s)
```

或直接计算：

```
>>> sum(range(1,101))
5050
```

4.3.2 break与continue语句

- 一旦break语句被执行，将使得break语句所属层次的循环提前结束；
- continue语句的作用是提前结束本次循环，忽略continue之后的所有语句，提前进入下一次循环。

4.3.2 break与continue语句

- 问题解决：计算小于100的最大素数。

```
for n in range(100, 1, -1):
    if n%2 == 0:
        continue
    for i in range(3, int(n**0.5)+1, 2):
        if n%i == 0:
            #结束内循环
            break
    else:
        print(n)
        #结束外循环
        break
```