第 4 章　选择结构与循环结构

4.1 条件表达式

- 绝大部分合法的 Python 表达式都可以作为条件表达式。
- 在选择和循环结构中，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。

4.1 条件表达式

```
>>> if 666:                    # 使用整数作为条件表达式
    print(9)

9
>>> a = [3, 2, 1]
>>> if a:                      # 使用列表作为条件表达式
   print(a)

[3, 2, 1]
```

4.1 条件表达式

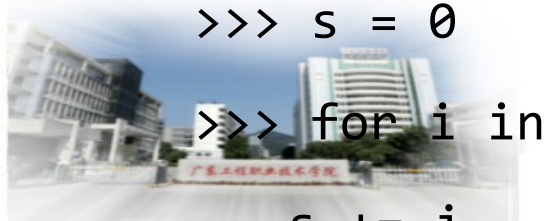
```
>>> a = []
>>> if a:                          # 空列表等价于 False
   print(a)
     else:
   print('empty')


empty
>>> i = s = 0
>>> while i <= 10:                 # 使用关系表达式作为条件表达式
```

4.1 条件表达式

```
>>> i = s = 0
>>> while True:          # 使用常量 True 作为条件表达式
    s += i
    i += 1
    if i > 10:          # 符合特定条件时使用 break 语句退出循环
        break

>>> print(s)
55
>>> s = 0
>>> for i in range(0, 11, 1):  # 遍历序列元素
```



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 、 or 、 not 分别表示逻辑与、逻辑或、逻辑非。对于 and 而言，必须两侧的表达式都等价于 True ，整个表达式才等价于 True 。
- 对于 or 而言，只要两侧的表达式中有一个等价于 True ，整个表达式就等价于 True ；对于 not 而言，如果后面的表达式等价于 False ，整个表达式就等价于 True 。
- 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。

4.1 条件表达式

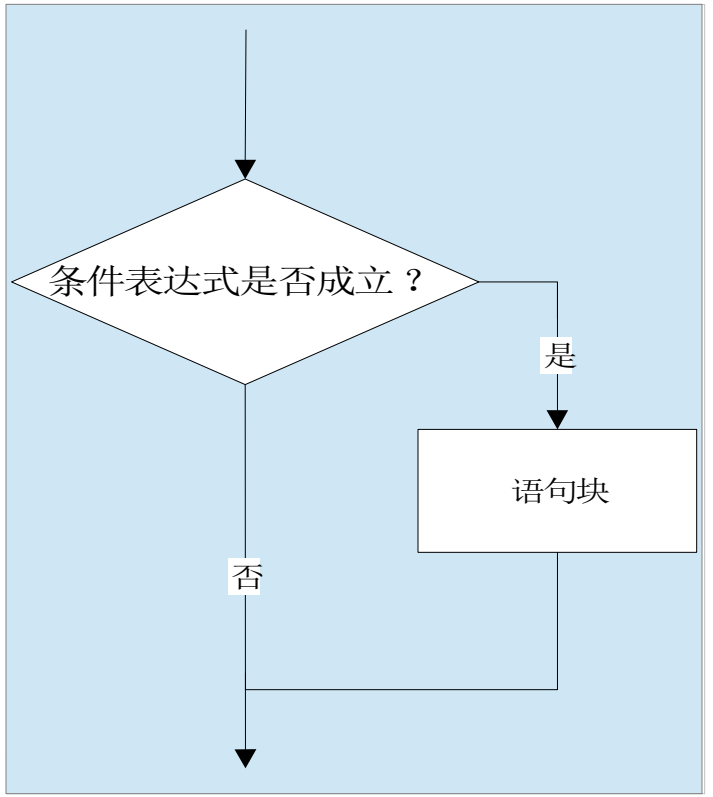
```
>>> 3 and 5                    # 整个表达式的值是最后一个计算的子表达式的值
5
>>> 3 or 5
3
>>> 0 and 5                    #0 等价于 False
0
>>> 0 or 5
5
>>> not [1, 2, 3]              # 非空列表等价于 True
False
>>> not {}                     # 空字典等价于 False
True
```

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、多分支选择结构以及嵌套的分支结构，也可以构造跳转表来实现类似的逻辑。
- 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。

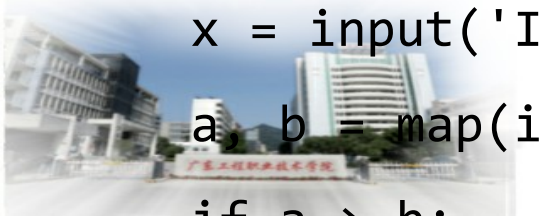
4.2.1 单分支选择结构

```
if 表达式：
    语句块
```



- **例** 4-1　编写程序，输入使用空格分隔的两个整数，然后按升序输出。

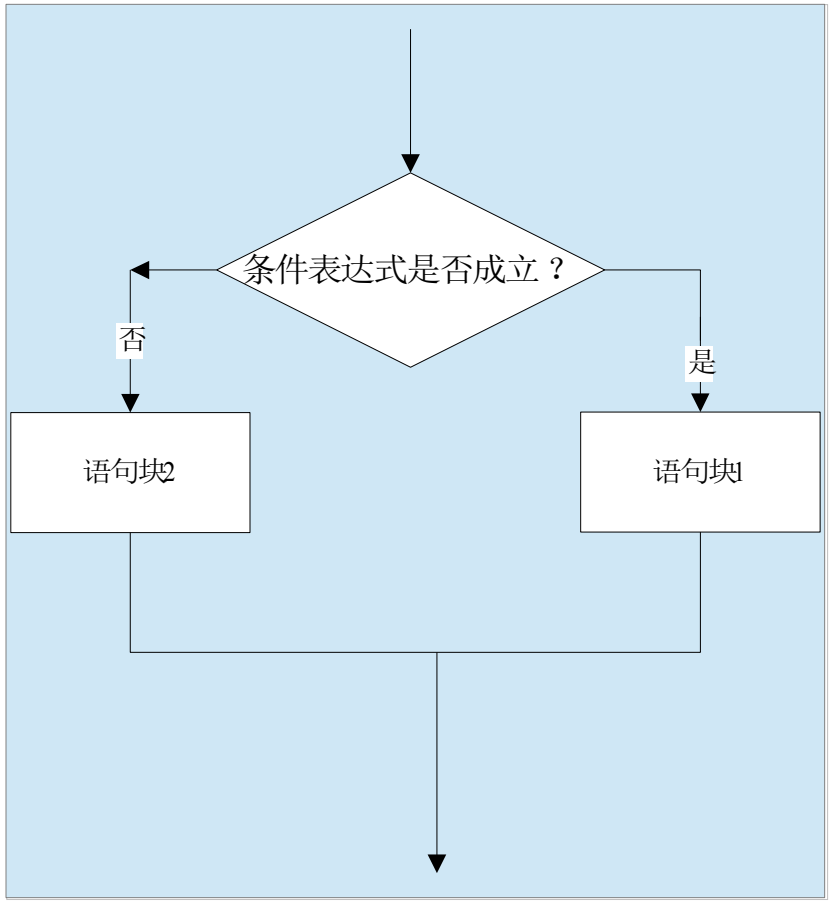
```
x = input('Input two number:')
a, b = map(int, x.split())
if a > b:
```



工于建构 成于创造

4.2.2 双分支选择结构

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



4.2.2 双分支选择结构

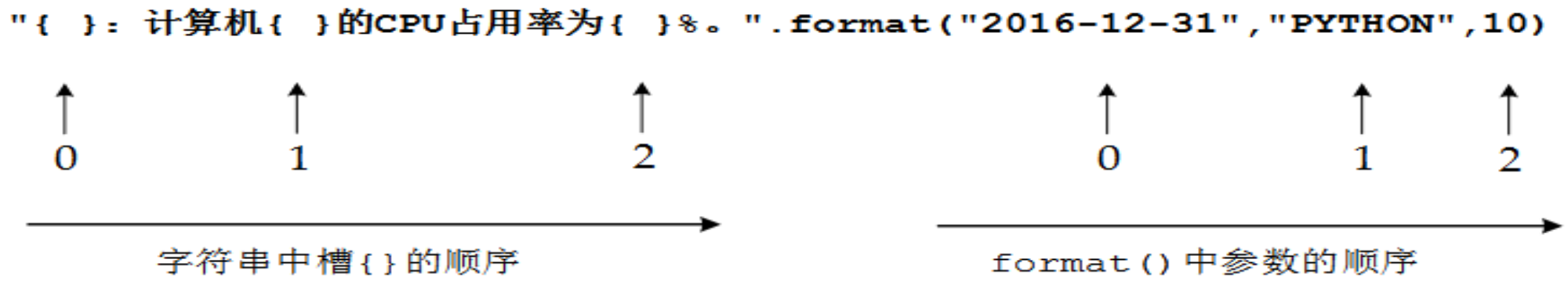
· **例 4-2** 求解鸡兔同笼问题。

```
jitu, tui = map(int, input('请输入鸡兔总数和腿总数：').split())
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')
```

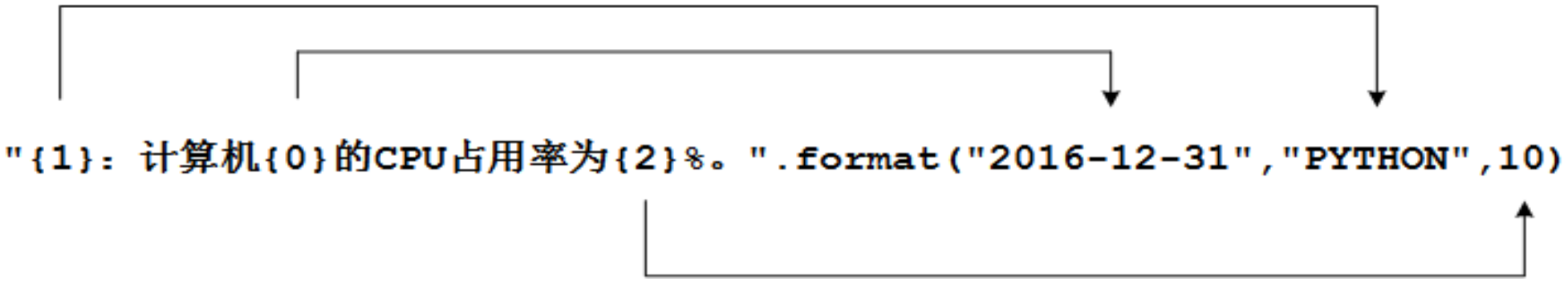
format() 方法的基本使用

字符串 format() 方法的基本使用格式是：

< 模板字符串 >.format(< 逗号分隔的参数 >)



format() 方法的基本使用



format() 方法的格式控制

- format() 方法中模板字符串的槽除了包括参数序号，还可以包括格式控制信息。此时，槽的内部样式如下：{< 参数序号 >: < 格式控制标记 >}
- 其中，格式控制标记用来控制参数显示时的格式 。格式控制标记包括： < 填充 >< 对齐 >< 宽度 >,<. 精度 >< 类型 >6 个字段，这些字段都是可选的，可以组合使用。

:	<填充>	<对齐>	<宽度>	,	<.精度>	<类型>
引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设定输出宽度	数字的千位分隔符 适用于整数和浮点数	浮点数小数部分的精度 或 字符串的最大输出长度	整数类型 b, c, d, o, x, X 浮点数类型 e, E, f, %

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

```
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

- **例** 4-3　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 'F'
```

4.2.4 选择结构的嵌套

```
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

- **例** 4-4　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。

4.3.1 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:
```

4.3.1 for 循环与 while 循环

- **例 4-5** 编写程序，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

· **例 4-6** 编写程序，打印九九乘法表。

```
for i in range(1, 10):
    for j in range(1, i+1):
        print('{0}*{1}={2}'.format(i,j,i*j), end='   ')
    print()                              # 打印空行，另起一行
```

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。

4.4 综合案例解析

- **例 4-8** 输入若干个成绩，求所有成绩的平均分。每输入一个成绩后询问是否继续输入下一个成绩，回答“yes”就继续输入下一个成绩，回答“no”就停止输入成绩。

4.4 综合案例解析

```
numbers = []
while True:
    x = input('请输入一个成绩：')
    # 异常处理结构，用来保证用户只能输入实数
    try:
        # 先把 x 转换成实数，然后追加到列表 numbers 尾部
        numbers.append(float(x))
    except:
        print('不是合法成绩')

    # 下面的循环用来限制用户只能输入任意大小写的“yes”或者“no”
    while True:
```

4.4 综合案例解析

- **例 4-10** 编写代码，输出由星号 * 组成的菱形图案，并且可以灵活控制图案的大小。

```
def main(n):
    for i in range(n):
        print((' * '*i).center(n*3))
    for i in range(n, 0, -1):
        print((' * '*i).center(n*3))
```

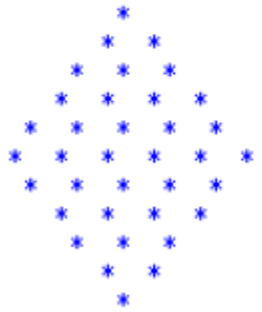


图 4-5 n=6 的运行效果

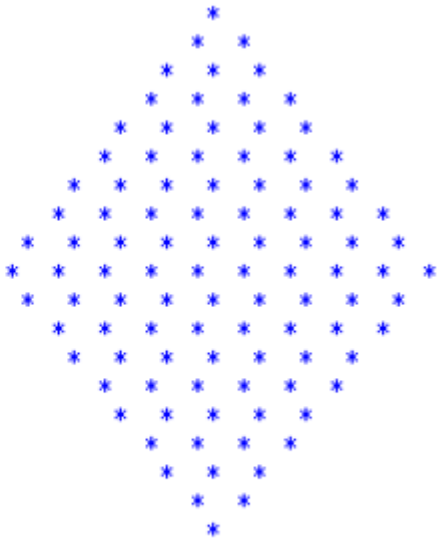


图 4-6 n=10 的运行效果

4.4 综合案例解析

- **例 4-14** 编写代码，模拟决赛现场最终成绩的计算过程。有至少 3 个评委，打分规则为删除最高分和最低分之后计算剩余分数的平均分。

```
while True:
    try:
        n = int(input(' 请输入评委人数： '))
        if n <= 2:
            print(' 评委人数太少，必须多于 2 个人。 ')
        else:
            break
    except:
```

4.4 综合案例解析

```
for i in range(n):
    # 这个 while 循环用来保证用户必须输入 0 到 100 之间的数字
    while True:
        try:
            score = input(' 请输入第 {0} 个评委的分数： '.format(i+1))
            # 把字符串转换为实数
            score = float(score)
            assert 0<=score<=100
            scores.append(score)
            # 如果数据合法，跳出 while 循环，继续输入下一个评委的分数
            break
```

4.4 综合案例解析

```
# 计算并删除最高分与最低分
highest = max(scores)
lowest = min(scores)
scores.remove(highest)
scores.remove(lowest)
finalScore = round(sum(scores)/len(scores),2)

formatter = '去掉一个最高分{0}\n去掉一个最低分{1}\n最后得分{2}'
print(formatter.format(highest, lowest, finalScore))
```