桂林理工大学
GUILIN UNIVERSITY OF TECHNOLOGY

第四章

程序控制结构

第四章 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。

第四章 程序控制结构

- 4.1条件表达式
- 4.2选择结构
- 4.3循环结构
- 4.4 精彩案例分析

4.1 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

（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

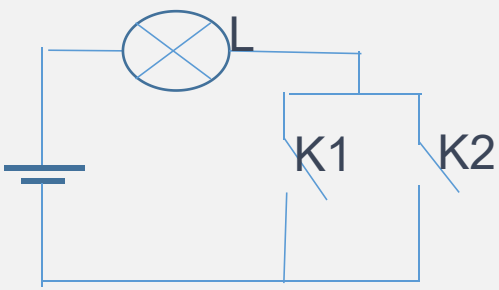
- 在Python语法中，条件表达式中不允许使用赋值运算符“=”，避免了误将关系运算符“==”写作赋值运算符“=”带来的麻烦。在条件表达式中使用赋值运算符“=”将抛出异常，提示语法错误。

```
>>> 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
```

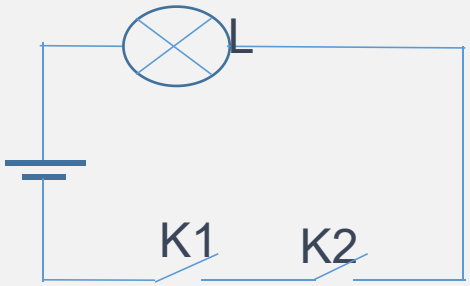
4.1 条件表达式

（2）逻辑运算符

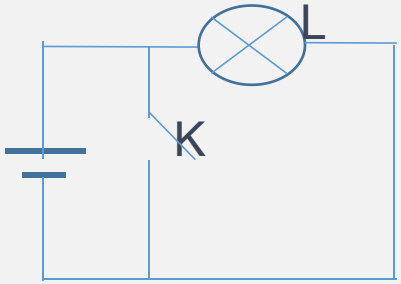
逻辑运算符and和or具有短路求值或惰性求值的特点，可能不会对所有表达式进行求值，而是只计算必须计算的表达式的值。



（1）or，并联电路



（2）and，串联电路



（3）not，短路

4.1 条件表达式

- 以“and”为例，对于表达式“表达式1 and 表达式2”而言，如果“表达式1”的值为“False”或其他等价值时，不论“表达式2”的值是什么，整个表达式的值都是“False”，丝毫不受“表达式2”的影响，因此“表达式2”不会被计算。
- 在设计包含多个条件的条件表达式时，如果能够大概预测不同条件失败的概率，并将多个条件根据“and”和“or”运算符的短路求值特性来组织顺序，可以大幅度提高程序运行效率。

4.1 条件表达式

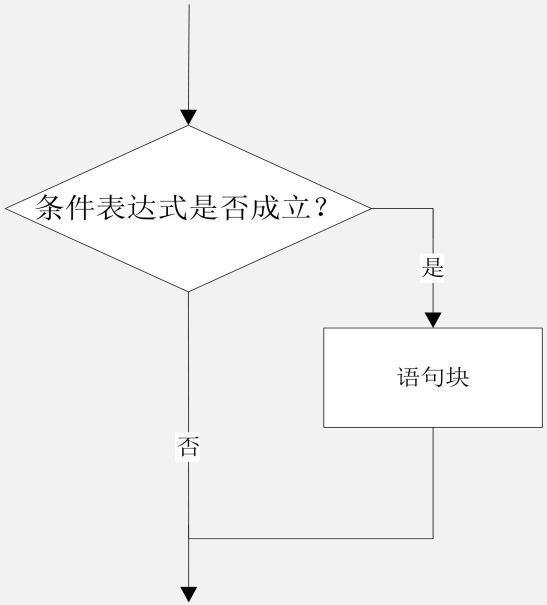
```
>>> 3 and 5
5
>>> 3 or 5
3
>>> 0 and 5
0
>>> 0 or 5
5
>>> not 3
False
>>> not 0
True
```

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

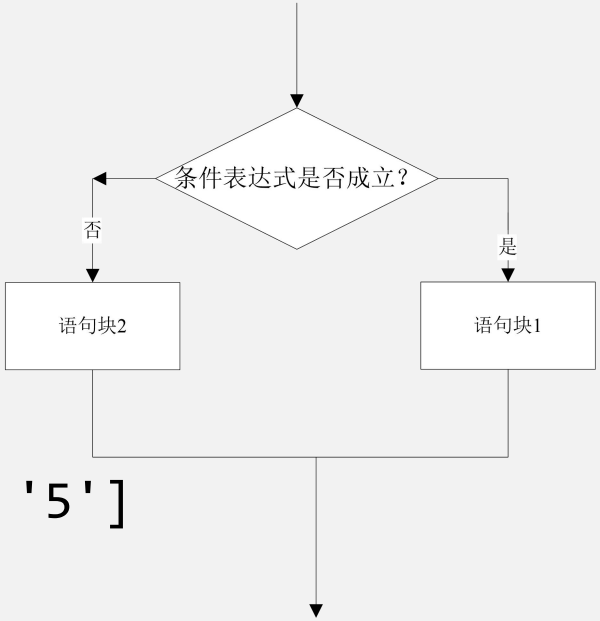
```
if 表达式:
    语句块
```




```
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)
```

4.2.2 双分支选择结构

```
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。

```
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')
```

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

- **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

```
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**：使用嵌套的循环结构打印九九乘法表。

```
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
```

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
```

4.4精彩案例赏析

- **示例4-1** 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精彩案例赏析

```
numbers = []                                    #使用列表存放临时数据
while True:
    x = input('请输入一个成绩: ')
    try:                                        #异常处理结构
        numbers.append(float(x))
    except:
        print('不是合法成绩')
    while True:
        flag = input('继续输入吗？（yes/no）')
        if flag.lower() not in ('yes', 'no'): #限定用户输入内容必须为yes或no
            print('只能输入yes或no')
        else:
            break
    if flag.lower()=='no':
        break

print(sum(numbers)/len(numbers))
```

4.4精彩案例赏析

- **示例4-2** 编写程序，判断今天是今年的第几天。

```
import time

date = time.localtime()                          #获取当前日期时间
year, month, day = date[:3]
day_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
if year%400==0 or (year%4==0 and year%100!=0):   #判断是否为闰年
    day_month[1] = 29
if month==1:
    print(day)
else:
    print(sum(day_month[:month-1])+day)
```

4.4精彩案例赏析

- **示例4-3** 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。

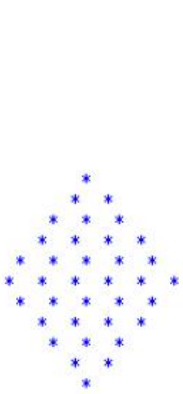


图 4-5　n=6 的运行效果

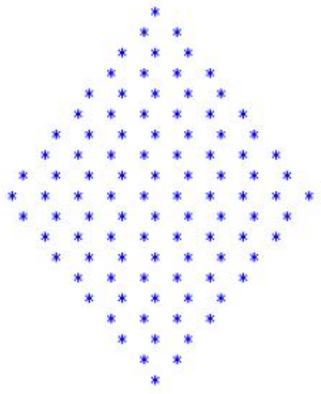


图 4-6　n=10 的运行效果

```
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))
```

4.4精彩案例赏析

- **示例4-4** 快速判断一个数是否为素数。

```
n = input("Input an integer:")
n = int(n)
if n == 2:
    print('Yes')
```

4.4精彩案例赏析

```
#偶数必然不是素数
elif n%2 == 0:
    print('No')
else:
    #大于5的素数必然出现在6的倍数两侧
    #因为6x+2、6x+3、6x+4肯定不是素数，假设x为大于1的自然数
    m = n % 6
    if m!=1 and m!=5:
        print('No')
    else:
        for i in range(3, int(n**0.5)+1, 2):
            if n%i == 0:
                print('No')
                break
        else:
            print('Yes')
```

4.4精彩案例赏析

- **示例4-5** 编写程序，计算组合数C(n,i)，即从n个元素中任选i个，有多少种选法。

根据组合数定义，需要计算3个数的阶乘，在很多编程语言中都很难直接使用整型变量表示大数的阶乘结果，虽然Python并不存在这个问题，但是计算大数的阶乘仍需要相当多的时间。本例提供另一种计算方法：以Cni(8,3)为例，按定义式展开如下，对于(5,8]区间的数，分子上出现一次而分母上没出现；(3,5]区间的数在分子、分母上各出现一次；[1,3]区间的数分子上出现一次而分母上出现两次。

$$C_8^3 = \frac{8!}{3!\times(8-3)!} = \frac{8\times7\times6\times\boldsymbol{5}\times\boldsymbol{4}\times\boldsymbol{3}\times\boldsymbol{2}\times\boldsymbol{1}}{3\times2\times1\times\boldsymbol{5}\times\boldsymbol{4}\times\boldsymbol{3}\times\boldsymbol{2}\times\boldsymbol{1}}$$

4.4精彩案例赏析

```
def Cni2(n, i):
    if not (isinstance(n,int) and isinstance(i,int) and n>=i):
        print('n and i must be integers and n must be larger than or equal to i.')
        return
    result = 1
    Min, Max = sorted((i,n-i))
    for i in range(n,0,-1):
        if i>Max:
            result *= i
        elif i<=Min:
            result /= i
    return result

print(Cni2(6,2))
```

4.4精彩案例赏析

- **示例4-6** 编写代码，模拟决赛现场最终成绩的计算过程。

```
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:
        pass

scores = []
```

4.4精彩案例赏析

```
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
        except:
            print('分数错误')
```


4.4精彩案例赏析

```
#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))
```