

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，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对象或其他能够使得bool()函数返回False的对象，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单个等于号“=”，**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单个等于号“=”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单个等于号
SyntaxError: invalid syntax
```

4.1 条件表达式

- 关系运算符具有惰性计算的特点，只计算必须计算的值，而不是计算整个表达式。

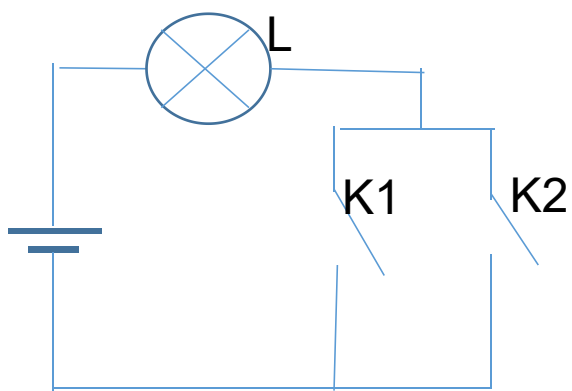
```
>>> 1 > 2 > xxx
```

```
False
```

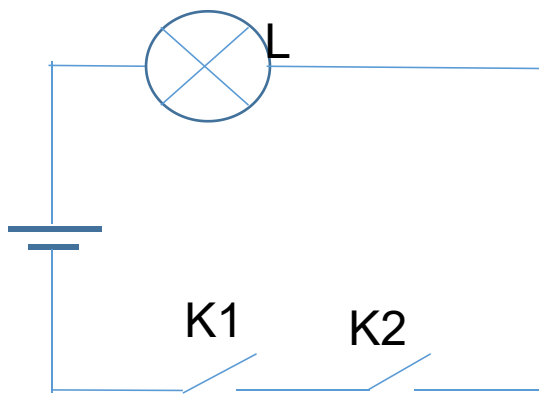
4.1 条件表达式

(2) 逻辑运算符

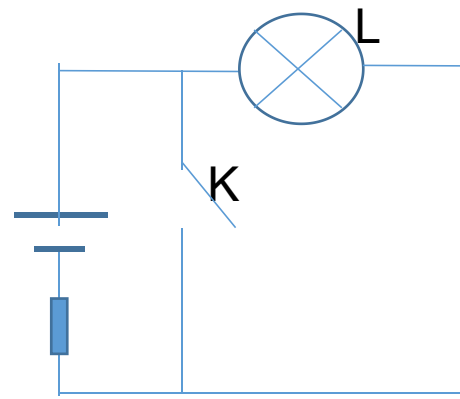
逻辑运算符and和or具有短路求值或惰性求值的特点，可能不会对所有表达式进行求值，而是只计算必须计算的表达式的值。



(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”的值等价于False时，不论“表达式2”的值是什么，整个表达式的值都等价于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
```

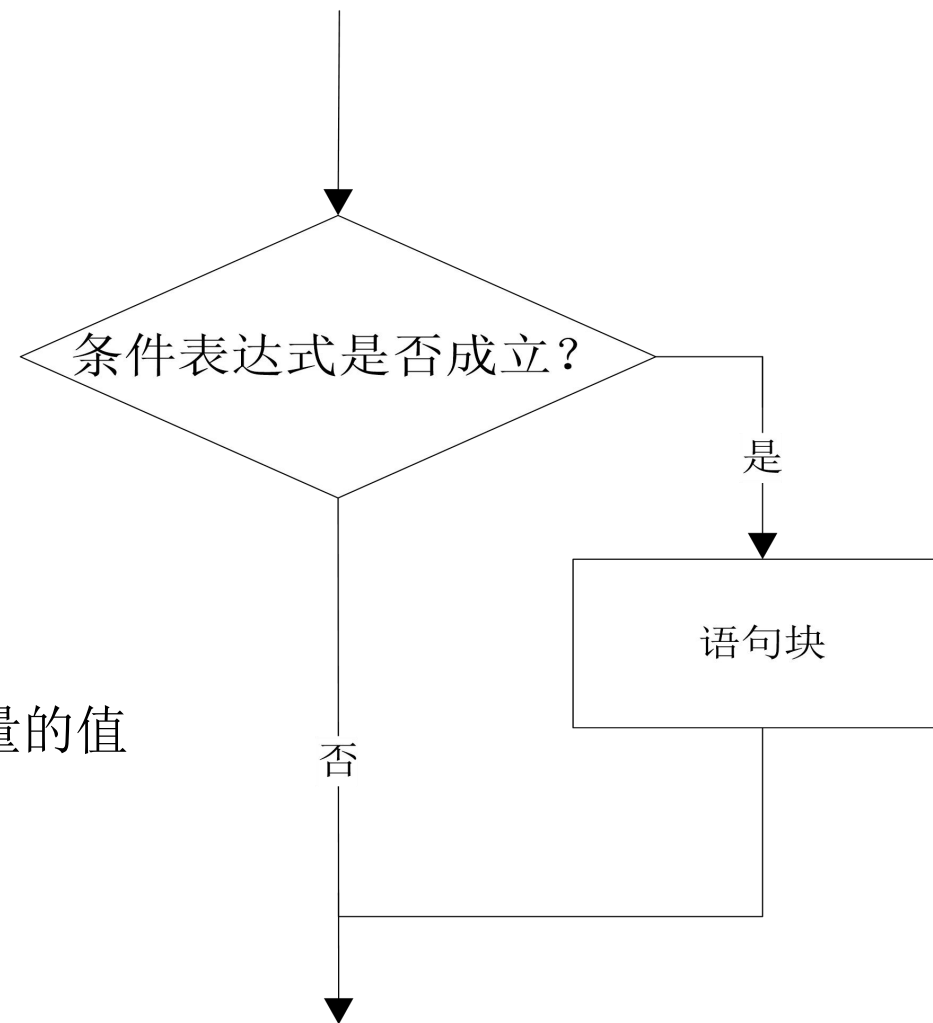
4.2 选择结构

- 选择结构根据特定的条件是否得到满足来决定下一步要执行的代码。
- 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

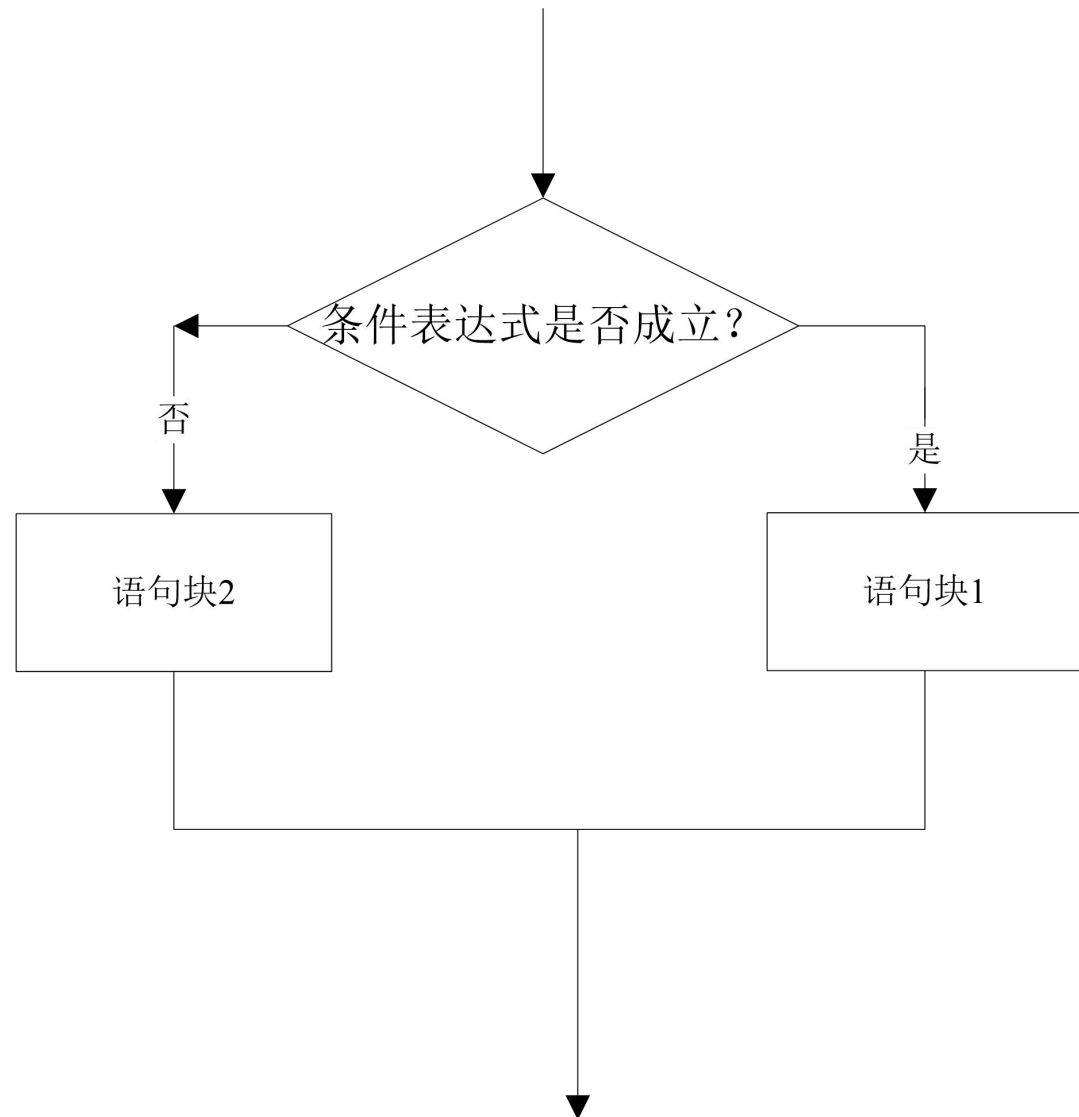
if 表达式:
 语句块

```
x = input('Input two integers:')  
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
```



4.2.2 双分支选择结构

- 问题解决：鸡兔同笼。 [code\4.2.2.py](#)

```
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 and 0<=tu<=jitu:
    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。

4.2.2 双分支选择结构

```
>>> print(6 if 5>3 else 5)
6
>>> b = 6 if 5>13 else 9
>>> b
9
>>> x = math.sqrt(9) if 5>3 else random.randint(1,100)
Traceback (most recent call last):
  File "<pyshell#140>", line 1, in <module>
    x = math.sqrt(9) if 5>3 else random.randint(1,100)
NameError: name 'math' is not defined
>>> import math
>>> x = math.sqrt(9) if 5>3 else random.randint(1,100)
>>> x = math.sqrt(9) if 2>3 else random.randint(1,100)
Traceback (most recent call last):
  File "<pyshell#143>", line 1, in <module>
    x = math.sqrt(9) if 2>3 else random.randint(1,100)
NameError: name 'random' is not defined
>>> import random
>>> x = math.sqrt(9) if 2>3 else random.randint(1,100)
```

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

- 应用：使用多分支选择结构将成绩从百分制变换到等级制。 [code\4.2.3.py](#)

```
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'  
  
print(func(90))
```

4.2.3 多分支选择结构

- Python 3.10新增关键字（只在特定场合中作为关键字，一般场合中可以作为变量名使用）`match`、`case`实现多分支选择结构。

```
>>> match 1+3:
...     case 5:
...         print(1)
...     case 3:
...         print(0)
...     case 4:
...         print(2)
...
...
2
>>> match 1+4:
...     case 5:
...         print(1)
...     case 3:
...         print(0)
...     case 4:
...         print(2)
...
...
1
```

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。

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

4.2.4 选择结构的嵌套

- 应用：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 = 'DCBAE'  
    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]  
  
print(func(90))
```

4.3 循环结构

- 循环结构用来重复执行代码，循环结构可以嵌套使用，也可以和选择结构嵌套使用来实现复杂的业务逻辑。
- Python主要有for循环和while循环两种形式。其中while循环常用于循环次数难以提前确定的情况，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可迭代对象中的元素。
- 对于带else子句的循环结构，如果循环因为条件表达式不成立或可迭代对象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` 循环变量 `in` 可迭代对象：

 循环体

[`else`:

`else`子句代码块]

4.3.1 for循环与while循环

- 应用：使用循环结构遍历并输出列表中的所有元素。 [code\4.3.1_1.py](#)

```
a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
```

```
for i, v in enumerate(a_list, start=1):
```

```
    print('列表的第', i, '个元素是: ', v)
```

4.3.1 for循环与while循环

- 应用：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。

[code\4.3.1_2.py](#)

```
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

- 应用：使用嵌套的循环结构打印九九乘法表。 [code\4.3.1_3.py](#)

```
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

- 应用：计算 $1+2+3+\dots+99+100$ 的结果。 [code\4.3.1_4.py](#)

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

或直接计算：

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

4.3.2 break与continue语句

- break和continue语句只能用在循环结构中。
- 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

- 应用：计算小于100的最大素数。 [code\4.3.2_1.py](#)

```
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
```

3.3.3 代码优化技巧

- 避免过多使用全局变量
- 尽量使用`from math import sin, cos`类似的方式导入标准库或扩展库对象
- 尽量减少循环循环中不必要的计算

3.3.3 代码优化技巧

- 优化前的代码:

```
digits = (1, 2, 3, 4)
```

```
for i in range(1000):
```

```
    result = []
```

```
    for i in digits:
```

```
        for j in digits:
```

```
            for k in digits:
```

```
                result.append(i*100+j*10+k)
```

3.3.3 代码优化技巧

- 优化后的代码:

```
for i in range(1000):  
    result = []  
    for i in digits:  
        i = i * 100  
        for j in digits:  
            j = j * 10  
            for k in digits:  
                result.append(i+j+k)
```

3.3.3 代码优化技巧

- 不推荐使用“模块名.对象名”的方式来访问。

```
import math
from time import time
```

```
start = time()
for i in range(100000000):
    math.sin(i)
print(time()-start)
```

10.982263326644897
9.380623579025269

```
start = time()
# 这种方式可以减少一次查表
loc_sin = math.sin
for i in range(100000000):
    loc_sin(i)
print(time()-start)
```


4.4 精彩案例赏析

- **例4-4** 快速判断一个数是否为素数。 [code\例4-4.py](#)

```
n = int(input('请输入一个大于1的自然数: '))
if n in (2,3):
    print('Yes')
# 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

- **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 $C_{8,3}$ 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。

$$C_8^3 = \frac{8!}{3! \times (8-3)!} = \frac{8 \times 7 \times 6 \times \textcolor{teal}{5} \times \textcolor{teal}{4} \times \textcolor{teal}{3} \times \textcolor{teal}{2} \times \textcolor{teal}{1}}{3 \times 2 \times 1 \times \textcolor{teal}{5} \times \textcolor{teal}{4} \times \textcolor{teal}{3} \times \textcolor{teal}{2} \times \textcolor{teal}{1}} = \frac{8 \times 7 \times 6}{3 \times 2 \times 1}$$

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 = result * i  
        elif i <= Min:  
            result = result // i  
    return result  
  
print(cni2(6,2))
```

[code\例4-5.py](#)