



布尔表达式

礼 欣

北京理工大学



布尔表达式

- 条件语句和循环语句都使用布尔表达式作为条件
- 布尔值为真或假，以False和True表示
- 前面经常使用布尔表达式比较两个值，如：`while x >= 0`



布尔操作符的引入

- 简单条件在复杂决策情况下存在一定缺陷
- 例如，确定两个点是否是在同一个位置，即是否有相同的x坐标和y坐标，下面是处理的代码片段：

```
if p1.getX() == p2.getX():  
    if p1.getY() == p2.getY():  
        # 两点相同  
    else:  
        # 两点不同  
else:  
    # 两点不同
```

过于复杂，Python
提供了更简单的布
尔操作符来构建表
达式



布尔操作符

- 布尔操作符： and , or 和 not
- 布尔运算符 and 和 or 用于组合两个布尔表达式，并产生一个布尔结果
 - `<expr> and <expr>`
 - `<expr> or <expr>`
- not 运算符是一个一元运算符，用来计算一个布尔表达式的反
 - `not <expr>`



布尔操作结果-真值表

| P | Q | P and Q |
|---|---|---------|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

| P | Q | P or Q |
|---|---|--------|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

| P | not P |
|---|-------|
| T | F |
| F | T |



布尔操作符

- 使用布尔运算符，可以建立任意复杂的布尔表达式
例如：`a or not b and c`
- Python中布尔操作符的优先级，从高分到低分依次是not、and最低是or。所以上面的达式等于如下这个带括号的版本：`(a or (not b) and c)`



布尔操作符

- 使用and操作符改进之前比较两个点相同的例子

```
if p1.getX() == p2.getX() and p2.getY() == p1.getY():  
    # 两点相同  
else:  
    # 两点不同
```

简单、清晰



壁球比赛计分例子

- 假设scoreA和scoreB代表两个壁球选手的分数
- 规则：只要一个选手达到了15分，本场比赛就结束。
即如下布尔表达式为真时比赛结束：

```
scoreA == 15 or scoreB == 15
```

- 可以构造循环条件，对游戏结束条件取反：

```
while not (scoreA == 15 or scoreB == 15):  
    # 比赛继续
```




壁球比赛计分例子

- a和b代表两个壁球选手的分数
- 规则1：只要一个选手达到了15分，本场比赛就结束；如果一方打了七分而另一方一分未得时，比赛也会结束

```
a == >>> (a>=15 or b>=15) and abs(a-b)>=2
```

- 规则2：需要一个团队赢得至少两分才算赢，即其中一个队已经达到了15分，且分数差异至少为2时比赛结束(排球)

```
(a >= 15 and a - b >= 2) or (b >= 15 and b - a >= 2)
```

- 等价于 `(a>=15 or b>=15) and abs(a-b)>=2`

abs函数返回表达式的绝对值



布尔代数

- 布尔表达式遵循特定的代数定律，这些规律被称为布尔逻辑或布尔代数
- 布尔代数规则

| Algebra↵ | Boolean algebra↵ |
|---------------|---|
| $a * 0 = 0$ ↵ | $a \text{ and } \text{false} == \text{false}$ ↵ |
| $a * 1 = a$ ↵ | $a \text{ and } \text{true} == a$ ↵ |
| $a + 0 = a$ ↵ | $a \text{ or } \text{false} == a$ ↵ |

- 当0和1对应false和true时
 - and与乘法相似
 - or与加法相似



布尔代数

- 任何值和true进行 “or” 操作都是真

```
a or true == true
```

- and和or 操作符都符合分配率：

```
a or (b and c) == (a or b) and (a or c)
a and (b or c) == (a and b) or (a and c)
```

- not操作符具有负负抵消的特性：.

```
not(not a) == a
```

- 布尔代数符合德摩根定律， not放进表达式后， and和or 运算符之间发生的变化：

```
not(a or b) == (not a) and (not b)
not(a and b) == (not a) or (not b)
```





布尔代数

■ 布尔代数的应用

```
while not (scoreA == 15 or scoreB == 15):  
    # 比赛继续
```

- 通过使用布尔代数,可以转换上面这个表达式。应用德摩根定律,其等同于下面这个表达式:

```
(not scoreA == 15) and (not scoreB == 15)
```

- 注意,当使用not的分配率时,or和and的转变。

```
while scoreA != 15 and scoreB != 15:  
    # 比赛继续
```



布尔表达式作为决策

- 在Python中，布尔表达式是很灵活的。
- 回顾交互式循环，只要用户响应一个“Y”程序就继续。为了让用户输入一个大写或小写，可以使用以下的循环：

```
while response[0] == "y" or response[0] == "Y":
```

- 初学者注意不要将上述表达式写成以下的形式

```
while response[0] == "y" or "Y":
```

- 其实这是一个无限循环。请思考为什么这里的条件表达式的值总为真？



布尔表达式作为决策

- Python的条件运算符（即`==`）总是在与一个bool类型的值进行比较！
- 布尔True和False来代表布尔值的真和假
- 对于数字（整型和浮点型）的零值被认为是false
- 任何非零值都是true
- bool类型仅仅是一个特殊的整数，可以通过计算表达式`True + True`的值来测试一下



布尔表达式作为决策

- 对于序列类型来说，一个空序列被解释为假，而任何非空序列被指示为真

```
>>> bool(0)
False
>>> bool(1)
True
>>> bool(32)
True
>>> bool("hello")
True
>>> bool("")
False
>>> bool([1,2,3])
True
>>> bool([])
False
>>>
```



布尔表达式-思考

- Python的布尔灵活性也扩展到了布尔运算符。下表总结了这些运算符的特性：

| operator↵ | operational definition↵ |
|------------------------|--|
| <code>x and y</code> ↵ | If <code>x</code> is false, return <code>x</code> . Otherwise, return <code>y</code> .↵ |
| <code>x or y</code> ↵ | If <code>x</code> is true, return <code>x</code> . Otherwise, return <code>y</code> .↵ |
| <code>not x</code> ↵ | If <code>x</code> is false, return <code>True</code> . Otherwise, return <code>False</code> .↵ |

- Python的布尔运算符是短路运算符。
- Python从左到右扫描表达式一旦知道结果，就立即返回True或False值。




布尔表达式-思考

```
while response[0] == "y" or response[0] == "Y":
```

```
while response[0] == "y" or "Y":
```

有何不同？

- 第二个表达式” Y”，它是一个非空的字符串，所以 Python 会永远把它解释为真。



布尔表达式-简洁表示

- 如果用户仅仅简单敲下回车键，可以使用方括号中的值作为默认值

```
ans = input("What flavor do you want [vanilla]: ")
if ans != "":
    flavor = ans
else:
    flavor = "vanilla"
```

- 可以简化如下：

```
ans = input("What flavor do you want [vanilla]: ")
if ans:
    flavor = ans
else:
    flavor = "vanilla"
```



布尔表达式-简洁表示

- 更简洁的表达形式

```
ans = input("What flavor do you want [vanilla]: ")  
flavor = ans or "vanilla"
```

- or操作符的定义保证它等价于if-else结构，

- 进一步简化：

```
flavor = input("What flavor do you want [vanilla]: ") or "vanilla"
```

