

Python程序设计

条件与循环

刘安

苏州大学，计算机科学与技术学院

<http://web.suda.edu.cn/anliu/>

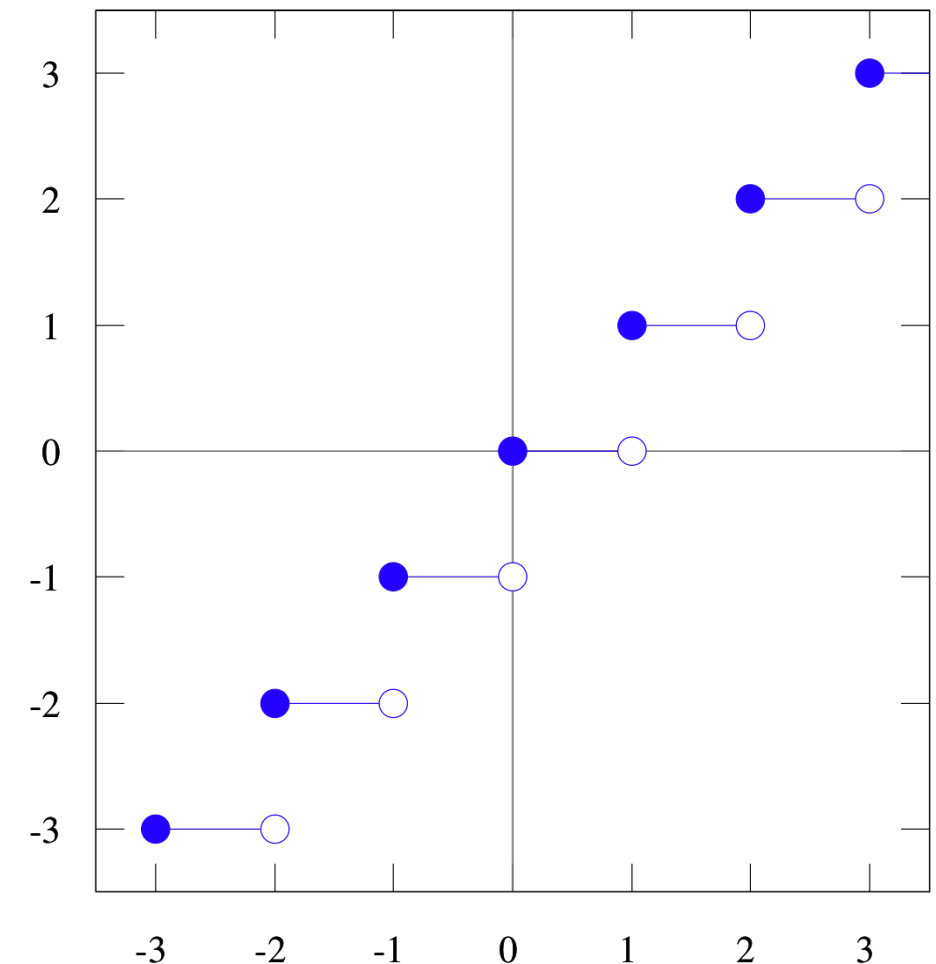
if-elif-else条件语句

- 1个if分支，后跟任意数量的elif分支（包括0个），后跟1个或者0个else分支，每个分支有一个对应的代码块
- 从上向下依次判断条件，如果为真，执行对应的代码块，否则判断下一个条件

```
1 if condition1:  
2     #code block 1  
3 elif condition2:  
4     #code block 2  
5 elif condition3:  
6     #code block 3  
7 else:  
8     #code block 4
```

if-elif-else条件语句实例

```
1 def my_floor(x):  
2     if x < 0:  
3         return math.floor(x)  
4     elif x < 1: # 0 <= x < 1  
5         return 0  
6     elif x < 2: # 1 <= x < 2  
7         return 1  
8     else: # x >= 2  
9         return math.floor(x)
```



嵌套的条件语句

- if-elif-else的各路分支里都可以包含新的if-elif-else语句

```
1 def my_floor(x):  
2     if x < 0:  
3         return math.floor(x)  
4     else: # x >= 0  
5         if x >= 2:  
6             return math.floor(x)  
7         elif x < 1: # 0 <= x < 1  
8             return 0  
9         else: # 1 <= x < 2  
10            return 1
```

对象的真值和假值

- 所有对象都有一个布尔值
- 非零数字或者非空对象都是真
- 0、空对象以及None都是假

```
>>> bool(None)
```

```
False
```

```
>>> bool(3.14)
```

```
True
```

```
>>> bool(0)
```

```
False
```

```
>>> bool('')
```

```
False
```

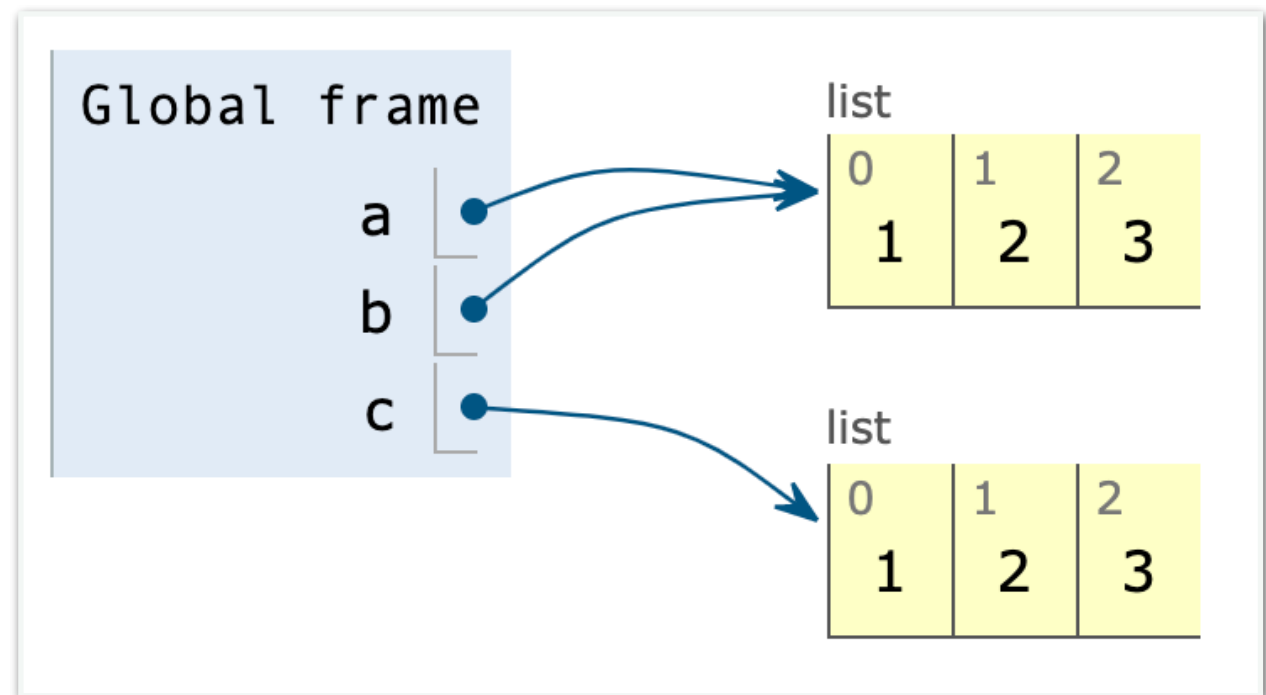
```
>>> bool([1, 2, 3])
```

```
True
```

对象比较：is和==

- ==比较的是两个对象的值是否相等（相等）
- is比较的是两个对象是否是同一个对象（相同）

```
>>> a = [1, 2, 3]
>>> b = a # a和b绑定到同一个列表
>>> c = [1, 2, 3] # c绑定到另一个列表
>>>
>>> a == b, a is b
(True, True)
>>>
>>> a == c, a is c
(True, False)
```



求列表的平均值

- 内置函数sum()返回列表所有元素的和
- 内置函数len()返回列表元素的个数

```
>>> L = [1, 2, 3, 4, 5]
```

```
>>> sum(L)
```

```
15
```

```
>>> len(L)
```

```
5
```

- 如果列表为空，约定平均值为0
- 不需要考虑列表中是否含有非数值类型

求列表的平均值

```
1 def avg_of_list(L):  
2     if L: ←  
3         return sum(L) / len(L)  
4     else:  
5         return 0
```

注意这里L直接作为条件
因为L有一个布尔值
而该值可以作为if条件测试的结果
如果为真，表示L为空
则执行if分支对应的代码块

```
>>> L = [1, 2, 3, 4, 5]  
>>> avg_of_list(L)  
3.0  
>>> L = []  
>>> avg_of_list(L)  
0
```



当L=[1, 2, 'hi']时，调用avg_of_list(L)会产生什么结果？

通过逻辑运算符表达复杂的条件

- **and, or, not**
 - 如果X和Y都为真, X **and** Y的布尔值为真
 - 如果X或Y为真, X **or** Y的布尔值为真
 - 如果X为假, **not** X的布尔值为真

```
>>> 1 < 3 and 3 < 5
```

```
True
```

```
>>> 1 < 3 or 3 > 5
```

```
True
```

```
>>> not 3 > 5
```

```
True
```

逻辑表达式的返回值

- `X and Y`和`X or Y`返回的不是左侧对象`X`就是右侧对象`Y`
- `X and Y` : 如果`X`为假, 返回`X`, 否则返回`Y`
- `X or Y` : 如果`X`为真, 返回`X`, 否则返回`Y`
- `not X` : 如果`X`为假, 返回`True`, 否则, 返回`False`

```
>>> 0 and []  
0
```

```
>>> 0 and 3  
0
```

```
>>> [1, 2] and 0  
0
```

```
>>> [1, 2] and 'hi'  
'hi'
```

```
>>> 0 or ''  
''
```

```
>>> '' or 3  
3
```

```
>>> [1, 2] or 0  
[1, 2]
```

```
>>> 'hi' or -5  
'hi'
```

```
>>> not []  
True
```

```
>>> not 'hi'  
False
```

在`X and Y`中, 如果`X`为假, 直接返回`X`, 而不再考虑`Y`, 这称为短路计算

if-else 三元表达式

- 第6行就是if-else三元表达式
 - 其效果等价于1~4行，但简洁很多

1	if X:	>>> x = 1 if 'hi' else 0
2	A = Y	>>> x
3	else:	1
4	A = Z	>>> x = 1 if [] else 0
5		>>> x
6	A = Y if X else Z	0

```
>>> L = [1, 2, 3, 4, 5]
>>> [1 if x % 2 == 1 else 0 for x in L]
[1, 0, 1, 0, 1]
```

while语句实现循环

- 循环：重复执行一个代码块
- **while** condition + 冒号 + 代码块（需要缩进）

```
1 #code before while  
2  
3 while condition:  
4     #code block  
5  
6 #code after while
```

- 只要第3行的condition为真，就会重复执行第4行的代码块
- 当第3行的condition为假时，跳过第4行的代码块，执行while语句之后的代码

求最大公约数

- 整数 m 和 n 的最大公约数 d 是能同时整除两者的最大的数
- 编写一个函数，接受两个整数，返回它们的最大公约数
- 穷举法：考虑 d 可能的取值范围
 - 上限： m 和 n 中较小者的绝对值，下限：1

```
1 def my_gcd(m, n):  
2     d = min(abs(m), abs(n))  
3     while True: # there must exist a gcd!  
4         if m % d == 0 and n % d == 0:  
5             return d  
6         d -= 1
```

求最大公约数

- 辗转相除法

- 如果 $m > n \neq 0$, 那么 $\text{gcd}(m, n) = \text{gcd}(n, m \% n)$
- 当 n 等于 0 时算法结束, 此时的 m 就是最大公约数
- $\text{gcd}(18, 12) = \text{gcd}(12, 6) = \text{gcd}(6, 0) = 6$

```
1 def my_gcd_v1(m, n):  
2     while n != 0:  
3         m, n = n, m % n # exchange 2 numbers  
4     return m
```



学习IPython中的`%timeit`命令, 使用该命令来测试
函数`my_gcd`和`my_gcd_v1`的执行时间, 比较两者性能上的差异

牛顿迭代法求平方根

1. 随意猜一个解`guess` (比如 $x/2$)
2. 评估解的质量
 - ① 如果不够好, 将`guess`更新成 $(guess + x/guess)/2$, 继续步骤2
 - ② 否则, 返回`guess`
- 什么叫“好”的解: `guess`的平方与 x 的差的绝对值小于一个很小的数, 比如 $1e-12$

```
1 def my_sqrt(x):  
2     guess = x / 2  
3     precision = 1e-12  
4     while abs(guess * guess - x) > precision:  
5         guess = (guess + x / guess) / 2  
6     return guess
```

```
>>> my_sqrt(2)  
1.414213562373095  
>>>  
>>> math.sqrt(2)  
1.4142135623730951
```

列表求和

- 编写一个函数，接受一个列表，返回该列表所有元素的和
(假设该列表中所有的元素都是数值)
- 列表的两个基本操作：索引和分片
- 列表本身的布尔值

```
1 def my_sum(L):  
2     total = 0  
3     while L: # L is not empty  
4         total = total + L[0]  
5         L = L[1:]  
6     return total
```

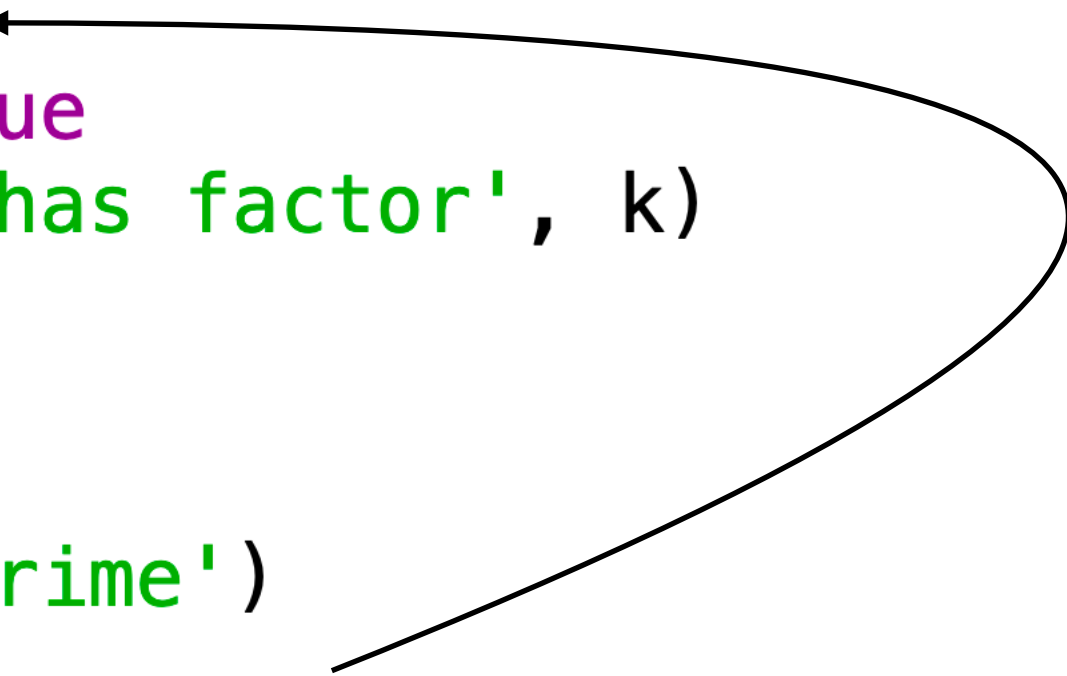
判定质数

- 一个大于1的自然数，除了1和它自身外，不能被其他自然数整除的数叫做质数，否则称为合数
- 编写一个函数，接受一个自然数，判断其是否是质数
 - 如果是，输出该数是质数的信息
 - 如果不是，输出该数的一个因子
- **break**语句：会导致执行流立刻从一个循环退出

判定质数

- break语句会导致执行流立刻从一个循环退出

```
1 def check_prime(n):  
2     found = False 标志变量found记录是否已经找到一个因子  
3     k = 2  
4     while k < n:  
5         if n % k == 0: ←  
6             found = True  
7             print(n, 'has factor', k)  
8             break  
9             k = k + 1  
10    if not found:  
11        print(n, 'is prime')
```



如果找到一个因子k可以整除n，就没有必要再去考虑更大的因子（第9行）
所以这里将标志变量found设置为True，输出相应信息，并使用break跳出while循环
最后通过标志变量的值，来决定是否输出n是质数的信息

帶有else分句的while语句

- 当且仅当没有执行break语句而正常退出循环时，才执行else分句

```
1 #code before while
2
3 while condition:
4     #code block
5     if condition:
6         break
7     #code block
8 else:
9     #code block
10
11 #code after while
```

带有else分句的while语句

- 当且仅当正常退出循环时（即没有执行break语句），才执行else分句

```
1 def check_prime_v1(n):  
2     k = 2  
3     while k < n:  
4         if n % k == 0:  
5             print(n, 'has factor', k)  
6             break  
7         k = k + 1  
8     else:  
9         print(n, 'is prime')
```



如果n的值为1，函数会输出什么样的信息？

break语句和continue语句

- **break**语句：执行流立刻从一个循环退出
- **continue**语句：执行流立刻跳到循环顶端（即判断条件）

```
1 def show_continue():  
2     x = 10  
3     while x:  
4         x = x - 1  
5         if x % 2 != 0:  
6             continue  
7         print(x, end = ' ')
```

```
>>> show_continue()  
8 6 4 2 0
```

使用for语句进行循环

- for语句用来遍历一个可迭代对象（比如range、列表、字符串、元组、字典、文件等），对于其中每一个元素e，重复执行for语句的代码块
- for语句也有一个可选的else分句，如果正常退出循环（即没有执行break语句），则执行该else分句

```
1 for e in object:
2     #####
3     # code block
4     if condition:
5         break
6     #####
7 else:
8     # code block
```

列表求和

- 使用for语句

```
1 def my_sum_v1(L):  
2     total = 0  
3     for e in L:  
4         total = total + e  
5     return total
```

- 使用while语句

```
1 def my_sum(L):  
2     total = 0  
3     while L: # L is not empty  
4         total = total + L[0]  
5         L = L[1:]  
6     return total
```

嵌套循环

- 现有两个列表A和B，对于A中的每个元素，在B中进行搜索，显示搜索结果

```
>>> A = [3.14, '10', 59, [1, 2]]
>>> B = [59, [1, 2]]
>>> for x in A:
    for y in B:
        if x == y:
            print(x, 'was found')
            break
    else:
        print(x, 'was not found')
```

```
3.14 was not found
10 was not found
59 was found
[1, 2] was found
```

```
>>> for x in A: # no nested loop
    if x in B:
        print(x, 'was found')
    else:
        print(x, 'was not found')
```

奇数对

- 编写一个函数，接受两个列表A和B，判断A中是否存在一个元素，其和B中一个元素的乘积是奇数。如果存在，返回True，否则返回False

```
1 def has_odd_pair(A, B):  
2     for x in A:  
3         for y in B:  
4             if (x * y) % 2 == 1:  
5                 return True  
6     return False
```

```
>>> A = [2, 0, 1, 9]  
>>> B = [2, 7, 1, 8]  
>>> has_odd_pair(A, B)  
True  
>>> C = [2, 4, 6, 8]  
>>> has_odd_pair(A, C)  
False
```

奇数对

- 乘积是奇数，意味着必须有2个奇数，也就是说，A中至少有一个奇数，同时B中至少有一个奇数，才会返回True

```
1 def has_odd_pair(A, B):  
2     for x in A:  
3         if x % 2 == 1: # 在A中找到奇数, 检查B  
4             break  
5     else: # A中没有奇数, 不用检查B  
6         return False  
7     for y in B:  
8         if y % 2 == 1: # B中找到奇数  
9             return True  
10    return False # B中没有奇数
```

生日悖论

- 在23个同学的班级中，50%的概率至少两个人生日相同
- 在60个同学的班级中，至少两个人生日相同的概率大于**99%**
- 编写一个函数模拟生日悖论
 - 在一次模拟试验中，随机生成n个同学的生日，如果至少有2个同学的生日相同，将该次试验结果标记为A，如果没有同学的生日相同，将该次试验结果标记为B
 - 进行t次模拟试验，统计结果为A的次数，记为x
 - 输出 x/t ，即至少两个人生日相同的概率



生日悖论

```
1 def simulate_birthday_paradox(n, t):  
2     cnt = 0  
3     for i in range(t): # t simulations  
4         dates = []  
5         for x in range(n): # the x-th student  
6             date = random.randint(1, 365)  
7             if date not in dates:  
8                 dates.append(date)  
9             else: # same birthday, then another simulation  
10                 cnt = cnt + 1  
11                 break  
12     return cnt / t
```

```
>>> simulate_birthday_paradox(23, 100000)  
0.50713  
>>> simulate_birthday_paradox(60, 100000)  
0.9939
```