**函数式编程**

**python把函数作为参数**

在2.1小节中，我们讲了高阶函数的概念，并编写了一个简单的高阶函数：

def add(x, y, f):

return f(x) + f(y)

如果传入abs作为参数f的值：

add(-5, 9, abs)

根据函数的定义，函数执行的代码实际上是：

abs(-5) + abs(9)

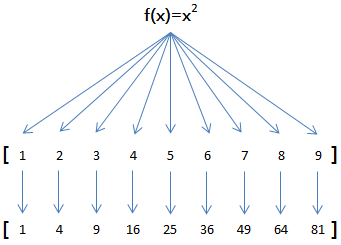
由于参数 x, y 和 f 都可以任意传入，如果 f 传入其他函数，就可以得到不同的返回值。

**python中map()函数**

map()是 Python 内置的高阶函数，它接收一个函数 f 和一个 list，并通过把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回。

例如，对于list [1, 2, 3, 4, 5, 6, 7, 8, 9]

如果希望把list的每个元素都作平方，就可以用map()函数：

[](http://img.mukewang.com/54c8a7e40001327303410245.png)

因此，我们只需要传入函数f(x)=x\*x，就可以利用map()函数完成这个计算：

def f(x):

return x\*x

print map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])

输出结果：

[1, 4, 9, 10, 25, 36, 49, 64, 81]

注意：map()函数不改变原有的 list，而是返回一个新的 list。

利用map()函数，可以把一个 list 转换为另一个 list，只需要传入转换函数。

由于list包含的元素可以是任何类型，因此，map() 不仅仅可以处理只包含数值的 list，事实上它可以处理包含任意类型的 list，只要传入的函数f可以处理这种数据类型。

**python中reduce()函数**

reduce()函数也是Python内置的一个高阶函数。reduce()函数接收的参数和 map()类似，一个函数 f，一个list，但行为和 map()不同，reduce()传入的函数 f 必须接收两个参数，reduce()对list的每个元素反复调用函数f，并返回最终结果值。

例如，编写一个f函数，接收x和y，返回x和y的和：

def f(x, y):

return x + y

调用 reduce(f, [1, 3, 5, 7, 9])时，reduce函数将做如下计算：

先计算头两个元素：f(1, 3)，结果为4；

再把结果和第3个元素计算：f(4, 5)，结果为9；

再把结果和第4个元素计算：f(9, 7)，结果为16；

再把结果和第5个元素计算：f(16, 9)，结果为25；

由于没有更多的元素了，计算结束，返回结果25。

上述计算实际上是对 list 的所有元素求和。虽然Python内置了求和函数sum()，但是，利用reduce()求和也很简单。

reduce()还可以接收第3个可选参数，作为计算的初始值。如果把初始值设为100，计算：

reduce(f, [1, 3, 5, 7, 9], 100)

结果将变为125，因为第一轮计算是：

计算初始值和第一个元素：f(100, 1)，结果为101。

**python中filter()函数**

filter()函数是 Python 内置的另一个有用的高阶函数，filter()函数接收一个函数 f 和一个list，这个函数 f 的作用是对每个元素进行判断，返回 True或 False，filter()根据判断结果自动过滤掉不符合条件的元素，返回由符合条件元素组成的新list。

例如，要从一个list [1, 4, 6, 7, 9, 12, 17]中删除偶数，保留奇数，首先，要编写一个判断奇数的函数：

def is\_odd(x):

return x % 2 == 1

然后，利用filter()过滤掉偶数：

filter(is\_odd, [1, 4, 6, 7, 9, 12, 17])

结果：[1, 7, 9, 17]

利用filter()，可以完成很多有用的功能，例如，删除 None 或者空字符串：

def is\_not\_empty(s):

return s and len(s.strip()) > 0

filter(is\_not\_empty, ['test', None, '', 'str', ' ', 'END'])

结果：['test', 'str', 'END']

注意: s.strip(rm) 删除 s 字符串中开头、结尾处的 rm 序列的字符。

当rm为空时，默认删除空白符（包括'\n', '\r', '\t', ' ')，如下：

a = ' 123'

a.strip()

结果： '123'

a='\t\t123\r\n'

a.strip()

结果：'123'

**python中自定义排序函数**

Python内置的 sorted()函数可对list进行排序：

>>>sorted([36, 5, 12, 9, 21])  
[5, 9, 12, 21, 36]

但 sorted()也是一个高阶函数，它可以接收一个比较函数来实现自定义排序，比较函数的定义是，传入两个待比较的元素 x, y，如果 x 应该排在 y 的前面，返回 -1，如果 x 应该排在 y 的后面，返回 1。如果 x 和 y 相等，返回 0。

因此，如果我们要实现倒序排序，只需要编写一个reversed\_cmp函数：

def reversed\_cmp(x, y):

if x > y:

return -1

if x < y:

return 1

return 0

这样，调用 sorted() 并传入 reversed\_cmp 就可以实现倒序排序：

>>> sorted([36, 5, 12, 9, 21], reversed\_cmp)

[36, 21, 12, 9, 5]

sorted()也可以对字符串进行排序，字符串默认按照ASCII大小来比较：

>>> sorted(['bob', 'about', 'Zoo', 'Credit'])

['Credit', 'Zoo', 'about', 'bob']

'Zoo'排在'about'之前是因为'Z'的ASCII码比'a'小。

**python中返回函数**

Python的函数不但可以返回int、str、list、dict等数据类型，还可以返回函数！

例如，定义一个函数 f()，我们让它返回一个函数 g，可以这样写：

def f():

print 'call f()...'

# 定义函数g:

def g():

print 'call g()...'

# 返回函数g:

return g

仔细观察上面的函数定义，我们在函数 f 内部又定义了一个函数 g。由于函数 g 也是一个对象，函数名 g 就是指向函数 g 的变量，所以，最外层函数 f 可以返回变量 g，也就是函数 g 本身。

调用函数 f，我们会得到 f 返回的一个函数：

>>> x = f() # 调用f()

call f()...

>>> x # 变量x是f()返回的函数：

<function g at 0x1037bf320>

>>> x() # x指向函数，因此可以调用

call g()... # 调用x()就是执行g()函数定义的代码

请注意区分返回函数和返回值：

def myabs():

return abs # 返回函数

def myabs2(x):

return abs(x) # 返回函数调用的结果，返回值是一个数值

返回函数可以把一些计算延迟执行。例如，如果定义一个普通的求和函数：

def calc\_sum(lst):

return sum(lst)

调用calc\_sum()函数时，将立刻计算并得到结果：

>>> calc\_sum([1, 2, 3, 4])

10

但是，如果返回一个函数，就可以“延迟计算”：

def calc\_sum(lst):

def lazy\_sum():

return sum(lst)

return lazy\_sum

# 调用calc\_sum()并没有计算出结果，而是返回函数:

>>> f = calc\_sum([1, 2, 3, 4])

>>> f

<function lazy\_sum at 0x1037bfaa0>

# 对返回的函数进行调用时，才计算出结果:

>>> f()

10

由于可以返回函数，我们在后续代码里就可以决定到底要不要调用该函数。

# [python中的set操作](http://blog.csdn.net/it_yuan/article/details/23170891)

**[python]** [view plain](http://blog.csdn.net/business122/article/details/7541486" \o "view plain" \t "http://blog.csdn.net/it_yuan/article/details/_blank)[copy](http://blog.csdn.net/business122/article/details/7541486" \o "copy" \t "http://blog.csdn.net/it_yuan/article/details/_blank)

1. python的set和其他语言类似, 是一个无序不重复元素集, 基本功能包括关系测试和消除重复元素. 集合对象还支持union(联合), intersection(交), difference(差)和sysmmetric difference(对称差集)等数学运算.
2. sets 支持 x **in** set, len(set),和 **for** x **in** set。作为一个无序的集合，sets不记录元素位置或者插入点。因此，sets不支持 indexing, slicing, 或其它类序列（sequence-like）的操作。
3. 下面来点简单的小例子说明把。
4. >>> x = set('spam')
5. >>> y = set(['h','a','m'])
6. >>> x, y
7. (set(['a', 'p', 's', 'm']), set(['a', 'h', 'm']))
8. 再来些小应用。
9. >>> x & y # 交集
10. set(['a', 'm'])
11. >>> x | y # 并集
12. set(['a', 'p', 's', 'h', 'm'])
13. >>> x - y # 差集
14. set(['p', 's'])
15. 记得以前个网友提问怎么去除海量列表里重复元素，用hash来解决也行，只不过感觉在性能上不是很高，用set解决还是很不错的，示例如下：
16. >>> a = [11,22,33,44,11,22]
17. >>> b = set(a)
18. >>> b
19. set([33, 11, 44, 22])
20. >>> c = [i **for** i **in** b]
21. >>> c
22. [33, 11, 44, 22]
23. 很酷把，几行就可以搞定。
24. 1.8　集合
25. 集合用于包含一组无序的对象。要创建集合，可使用set()函数并像下面这样提供一系列的项：
26. s = set([3,5,9,10])      #创建一个数值集合
27. t = set("Hello")         #创建一个唯一字符的集合
28. 与列表和元组不同，集合是无序的，也无法通过数字进行索引。此外，集合中的元素不能重复。例如，如果检查前面代码中t集合的值，结果会是：
29. >>> t
30. set(['H', 'e', 'l', 'o'])
31. 注意只出现了一个'l'。
32. 集合支持一系列标准操作，包括并集、交集、差集和对称差集，例如：
33. a = t | s          # t 和 s的并集
34. b = t & s          # t 和 s的交集
35. c = t – s          # 求差集（项在t中，但不在s中）
36. d = t ^ s          # 对称差集（项在t或s中，但不会同时出现在二者中）
37. 基本操作：
38. t.add('x')            # 添加一项
39. s.update([10,37,42])  # 在s中添加多项
40. 使用remove()可以删除一项：
41. t.remove('H')
42. len(s)
43. set 的长度
44. x **in** s
45. 测试 x 是否是 s 的成员
46. x **not** **in** s
47. 测试 x 是否不是 s 的成员
48. s.issubset(t)
49. s <= t
50. 测试是否 s 中的每一个元素都在 t 中
51. s.issuperset(t)
52. s >= t
53. 测试是否 t 中的每一个元素都在 s 中
54. s.union(t)
55. s | t
56. 返回一个新的 set 包含 s 和 t 中的每一个元素
57. s.intersection(t)
58. s & t
59. 返回一个新的 set 包含 s 和 t 中的公共元素
60. s.difference(t)
61. s - t
62. 返回一个新的 set 包含 s 中有但是 t 中没有的元素
63. s.symmetric\_difference(t)
64. s ^ t
65. 返回一个新的 set 包含 s 和 t 中不重复的元素
66. s.copy()
67. 返回 set “s”的一个浅复制
68. 请注意：union(), intersection(), difference() 和 symmetric\_difference() 的非运算符（non-operator，就是形如 s.union()这样的）版本将会接受任何 iterable 作为参数。相反，它们的运算符版本（operator based counterparts）要求参数必须是 sets。这样可以避免潜在的错误，如：为了更可读而使用 set('abc') & 'cbs' 来替代 set('abc').intersection('cbs')。从 2.3.1 版本中做的更改：以前所有参数都必须是 sets。
69. 另外，Set 和 ImmutableSet 两者都支持 set 与 set 之间的比较。两个 sets 在也只有在这种情况下是相等的：每一个 set 中的元素都是另一个中的元素（二者互为subset）。一个 set 比另一个 set 小，只有在第一个 set 是第二个 set 的 subset 时（是一个 subset，但是并不相等）。一个 set 比另一个 set 打，只有在第一个 set 是第二个 set 的 superset 时（是一个 superset，但是并不相等）。
70. 子 set 和相等比较并不产生完整的排序功能。例如：任意两个 sets 都不相等也不互为子 set，因此以下的运算都会返回 False：a<b, a==b, 或者a>b。因此，sets 不提供 \_\_cmp\_\_ 方法。
71. 因为 sets 只定义了部分排序功能（subset 关系），list.sort() 方法的输出对于 sets 的列表没有定义。
72. 运算符
73. 运算结果
74. hash(s)
75. 返回 s 的 hash 值
76. 下面这个表列出了对于 Set 可用二对于 ImmutableSet 不可用的运算：
77. 运算符（voperator）
78. 等价于
79. 运算结果
80. s.update(t)
81. s |= t
82. 返回增加了 set “t”中元素后的 set “s”
83. s.intersection\_update(t)
84. s &= t
85. 返回只保留含有 set “t”中元素的 set “s”
86. s.difference\_update(t)
87. s -= t
88. 返回删除了 set “t”中含有的元素后的 set “s”
89. s.symmetric\_difference\_update(t)
90. s ^= t
91. 返回含有 set “t”或者 set “s”中有而不是两者都有的元素的 set “s”
92. s.add(x)
93. 向 set “s”中增加元素 x
94. s.remove(x)
95. 从 set “s”中删除元素 x, 如果不存在则引发 KeyError
96. s.discard(x)
97. 如果在 set “s”中存在元素 x, 则删除
98. s.pop()
99. 删除并且返回 set “s”中的一个不确定的元素, 如果为空则引发 KeyError
100. s.clear()
101. 删除 set “s”中的所有元素
102. 请注意：非运算符版本的 update(), intersection\_update(), difference\_update()和symmetric\_difference\_update()将会接受任意 iterable 作为参数。从 2.3.1 版本做的更改：以前所有参数都必须是 sets。
103. 还请注意：这个模块还包含一个 union\_update() 方法，它是 update() 方法的一个别名。包含这个方法是为了向后兼容。程序员们应该多使用 update() 方法，因为这个方法也被内置的 set() 和 frozenset() 类型支持。