链滴

python 笔记 1

作者：zhuhonglin

原文链接：https://ld246.com/article/1494639019578

来源网站：链滴

闭包

内层函数引用了外层函数的变量（参数也算变量），然后返回内层函数的情况，称为闭包（Closure）

如:

```
def calc_sum(lst):
    def lazy_sum():
        return sum(lst)
    return lazy_sum
```

闭包的特点是返回的函数还引用了外层函数的局部变量，所以，要正确使用闭包，就要确保引用的局变量在函数返回后不能变

```
def count():
    fs = []
    for i in range(1, 4):
        def f(j):
            def g():
                return j*j
            return g
        fs.append(f(i))
    return fs
```

这个例子里就使用了闭包：count() 返回的是 一个list，这个list里面有三个元素，三个指向不同函数引用.

f(i)就是g，这三个g也就是list的元素。

接下来想得到值：

```
f = count()
f[0]() ==> 1
f[1]() ==> 4
f[2]() ==> 9
```

匿名函数

匿名函数使用lambda关键字

如：

```
lambda x: x * x  #冒号前面的x表示参数，后面表示返回值
#等同于：
def f(x):
    return x * x
```

匿名函数的限制：只能有一个表达式，返回值就是该表达式的结果（也就是上面的 x * x）

装饰器decorator

装饰器接受一个函数作为参数，返回一个新函数，本质上是一个高阶函数，使用它修饰原函数

如：

```
def new_f(f):
    def fn(x):
        print 'call ' + f.__name__ + '()...'
        return f(x)
    return fn
def f(n):
    return reduce(lambda x,y: x*y, range(1, n+1))
f = new_f(f)
print f(10)
==>call f()...
==>某个值
```

@是python内置的语法，简化调用

如

```
@new_f
def f(n):
    return reduce(lambda x,y: x*y, range(1, n+1))
#等价于:
def f(n):
    return reduce(lambda x,y: x*y, range(1, n+1))
f = new_f(f)
```

模块和包

```
import util.test
print util.test.f() #util包,test模块,f函数
```

反应在win本地，包就是文件夹，模块就是.py文件，每一个包下必须有一个_
nit_.py文件

函数

map()函数

map(f,[])

对list中的每一项使用f函数,f本身接受一个参数，返回一个新的list

---

reduce()函数

reduce(f,[],初始值)

f必须接受两个参数，对list每个元素调用，返回一个值

例如：

def f(x,y)

```
return x+y
```

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

过程为:

f(1,3)-->f(4,5)-->f(9,7)-->f(16,9)-->25

如果有初始值(可以没有)，则从初始值开始进行，上面假如初始值100，则结果为125

---

filter()函数

filter(f,[])

f返回true或false，返回符合条件的值，组成新的list

---

sorted()函数

sorted([],f)

对[]中的元素进行排序

f可以没有，如果没有f，默认从小到大排序

f(x,y) x和y是两个待比较的元素，x前y后，需要返回负数；反之需要返回正数。相等返回0

sorted函数返回一个新的list

类

```
class A(object):      #a,b,c任意，self必须是_init_的第一个参数(别的关键字也行)
    address = 'earth'  #类属性（类似java类静态成员）
    def __init__ (self, a, b, c, **kw):
        self.a = a
        self.b = b
        self.c = c
        for k, v in kw.iteritems():
            setattr(self, k, v)
```

```
        #setattr:向self所指的对象添加 属性名k 和 属性值v。
        #delattr(self, k)删除元素，k为属性名。与上面的相对应
    @classmethod   #类方法声明，Fn为类方法。
    def Fn(cls){
        return cls.address
    }
print A.address #不管是在类内还是类外使用，使用时必须是 类名.类属性 的方式
hhh = A('aaa','bbb','ccc', d = 'ddd')
```

1. __双下划线开头的属性，匿名
2. 但是__xxx__可以被外部访问，这种形式是特殊属性。
3. 实例属性和类属性重名时，实例属性优先级高
4. 类方法只能访问类属性,参数只代表类，别的关键字也行

继承

```
class A(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
class B(A):
    def __init__(self, x, y, z):
        super(B, self).__init__(x, y) #使用super初始化父类
        self.z = z
```

在super中，第一个参数表示寻找这个参数的父类，第二个参数代指父类，调用其构造方法，后面的参表里不用写self，已经**隐式传递**

"多态"

python本身不支持多态，但可以实现差不多的目的

```
class A:
    def p(self):
        print "A"


class B(A):
    def p(self):
        print "B"
def p1(obj):
    obj.p()
a = A()
b = B()
p1(a)
p2(b)
==>A
==>B
```

动态语言不检查参数的类型，所以这样做成立，实现了类似的多态

多重继承

这里主要想到了c++多重继承的时候，二义性的问题。python2.7和python3都可以用"广度搜索"解决

```
class C(A,B) #省略A,B类
c1 = C
c1.f()  #若A,B都有f函数，则"从左到右"，调用A的f。如果A，B都没有，则开始找A，B的上层。
```