高阶函数

函数

函数本身也可以赋值给变量,即:变量可以指向函数。

```
In [30]: abs(-10)
Out[30]: 10

In [31]: abs
Out[31]: <function abs>

In [32]: x = abs(-10)

In [33]: print x
10

In [34]: f = abs

In [35]: f
Out[35]: <function abs>

In [36]: f(-10)
Out[36]: 10
```

函数

函数名其实就是指向函数的变量!

```
In [37]: abs(-10)
Out[37]: 10

In [38]: abs = 1

In [39]: abs(-10)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-c432e3f1fd6c> in <module>()
----> 1 abs(-10)

TypeError: 'int' object is not callable
```

上述操作发现：abs为函数名，给abs=1重新赋值后，abs已不是函数，而是一个整数。

高阶函数

变量可以指向函数,函数的参数能接收变量,那么一个函数就可以接收另一个函数作为参数,这种函数就称之为高阶函数。

```
In [5]: def hfunc(x,y,f):
   ...:     print f(x),f(y)
   ...:

In [6]: hfunc(2,-10,abs)
2 10
```

map函数

map() 函数接收两个参数,一个是函数,一个是序列, map 将传入的函数依次作用到序列的每个元素,并把结果作为新的 list 返回。

```
In [10]: def f(x):
    ....:     return x*x
    ....:

In [11]: map(f,range(10))
Out[11]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

不需要 map() 函数,写一个循环,也可以计算出结果.

map函数

map() 作为高阶函数,把运算规则抽象了.

练习： 把这list列表中的所有数字转为字符串；([1,2,3]---['1','2','3'])

reduce函数

reduce 把一个函数作用在一个序列[x1, x2, x3...]上,这个函数必须接收两个参数,reduce 把结果继续和序列的下一个元素做累积计算。

```
In [20]: def add(x,y):
   ....:     return x+y
   ....:

In [21]: reduce(add,[2,3,4])
Out[21]: 9

In [22]: def fn(x,y):
   ....:     return x * 10 + y
   ....:

In [23]: reduce(fn,[1,2,5,6])
Out[23]: 1256
```

reduce函数

综合编程：写出把 str 转换为 int 的函数(eg:'12345'--12345)

reduce函数

```
In [24]: def fn(x,y):
    ....:     return x * 10 + y
    ....:

In [25]: def char2num(s):
    ....:     return {'0':0, '1':1, '2':2, '3':3, '4':4, '5':5,'6':6, '7':7, '8':
8,'9':9}[s]
    ....:

In [26]: reduce(fn, map(char2num,'13579'))
Out[26]: 13579
```

map/reduce练习题

•利用 map() 函数,把用户输入的不规范的英文名字,变为首字母大写,其他小写的规范名字。输入: ['adam', 'LISA', 'barT'] ,输出: ['Adam', 'Lisa', 'Bart'] 。

•Python 提供的 sum() 函数可以接受一个 list 并求和,请编写一个 prod() 函数,可以接受一个 list 并利用 reduce() 求积。

filter函数

filter() 也接收一个函数和一个序列。和 map() 不同的时，filter() 把传入的函数依次作用于每个元素,然后根据返回值是 True 还是 False 决定保留还是丢弃该元素。

filter函数

在一个 list 中,删掉偶数,只保留奇数:

```
In [29]: def isodd(n):
    ....:     return n % 2 == 1
    ....:

In [30]: filter(isodd,range(10))
Out[30]: [1, 3, 5, 7, 9]
```

filter函数

- 把一个序列中的空字符串删请尝试用 filter() ;
- 用 filter()删除 1~100 的素数;

sorted函数

- 排序也是在程序中经常用到的算法。 无论使用冒泡排序还是快速排序,排序的核心是比较两个元素的大小。通常规定如下:

    x < y, return -1

    x == y, return 0

    x > y, return 1

sorted函数

- python内置的 sorted() 函数就可以对 list 进行排序;
- 如果要倒序排序呢？
- 如果要对字符串进行排序呢？

sorted函数

倒序排序

```
In [59]: sorted([23,1,67])
Out[59]: [1, 23, 67]

In [60]: def reversed_cmp(x,y):
   ....:     if x > y:
   ....:         return -1
   ....:     if x < y:
   ....:         return 1
   ....:     return 0
   ....:

In [61]: sorted([23,1,67],reverse)
reverse=      reversed      reversed_cmp

In [61]: sorted([23,1,67],reversed_cmp)
Out[61]: [67, 23, 1]
```

高阶函数的抽象能力非常强大,而且核心代码可以保持得非常简洁。

函数作为返回值

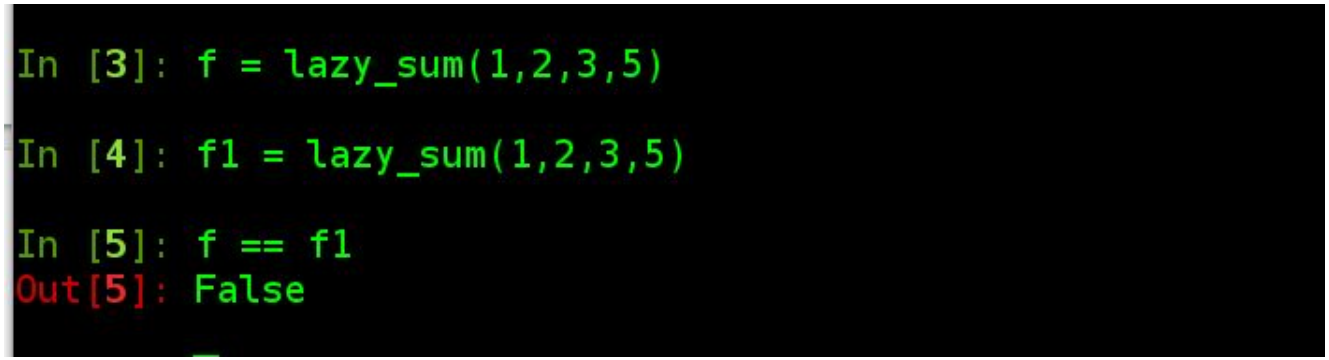
高阶函数除了可以接受函数作为参数外,还可以把函数作为结果值返回。

```
def lazy_sum(*args):                    1
    def sum():                          5
        ax = 0                          6
        for n in args:                  7
            ax = ax + n                 8
        return ax                       9
    return sum                          3

f = lazy_sum(1,2,3,4)                   2
          #调用 lazy_sum() 时,返回的并不是求和结果,而是求和函数
f()                                     4,10
          #调用函数 f 时,才真正计算求和的结果
```

函数作为返回值

调用 lazy_sum() 时,每次调用都会返回一个新的函数,即使传入相同的参数



匿名函数

- 当我们在传入函数时,有些时候,不需要显式地定义函数,直接传入匿名函数更方便。
- 关键字 lambda 表示匿名函数,冒号前面的 x 表示函数参数

```
In [6]: map(lambda x: x*x, [1,2,3,4])
Out[6]: [1, 4, 9, 16]

In [7]: def func(x):
   ...:     return x*x
   ...:

In [8]: map(func,[1,2,3,4])
Out[8]: [1, 4, 9, 16]
```

匿名函数

- 匿名函数有只能有一个表达式,不用写 return ,返回值就是该表达式的结果。
- 因为匿名函数没有名字,不必担心函数名冲突。 此外,匿名函数也是一个函数对象,也可以把匿名函数赋值给一个变量,再利用变量来调用该函数;

```
In [9]: f = lambda x : x*x

In [10]: f
Out[10]: <function __main__.<lambda>>

In [11]: f(2)
Out[11]: 4
```

匿名函数

- 也可以把匿名函数作为返回值返回

```
n [13]: def build(x,y):return lambda:x * x + y * y

n [14]: build(1,2)
ut[14]: <function __main__.<lambda>>

n [15]: a = build(1,2)

n [16]: a()
ut[16]: 5
```

装饰器

- 装饰器就是用来装饰函数。
  - 想要增强原有函数的功能;
  - 但不希望修改now()函数的定义;
  - 在代码运行期间动态增加功能的方式;

装饰器

- 定义的装饰器实质是返回函数的高阶函数。(试试下面的装饰器)

```
import time
def timeIt(func):
        def warp(arg):
                start = datetime.datetime.now()
                func(arg)
                end = datetime.datetime.now()
                cost = end - start
                print "execute %s spend %s" % (func.__name__,cost.total_seconds())
        return warp
@timeIt # 这里是 python 提供的一个语法糖
def func(arg):
        time.sleep(arg)
func(3)
```

over !