递归函数 匿名函数 函数式编程(map filter reduce)

函数的设计概念:

高内聚 低耦合

实现低耦合

1. 为力求让函数独立于她外部的其他代码 对于函数输入使用参数，输出使用 Return 语句

2.只有在真正必要的情况下才使用全局变量

3.避免直接改变在另一个模块文件中的变量

实现高内聚

1.每一个函数都应该有一个单一的目标 在设计完美的情况下 一个函数应该只做一件事 并且这件事可以用一个简单的说明语句来总结

2.一个函数中的代码行数应该控制在一个合理范围 如果一个函数的代码需要翻几页才能看完，此时应考虑是否可以对它进行再细分

Python 支持递归函数

什么是递归函数?

在一个函数内部可以调用其他函数 ，如果一个函数在自身内部调用自身本身， 这种直接或间接的调用自身以循环的函数 它允许程序遍历拥有任意的 不可预知的形状的结构

1. 递归求和

我们虽然可以使用 python 内置的 sum 函数来进行数字列表求和 也可以通过递归函数的形式来做

```
def calc(x):
    print x
    if not x:
        return 0
    else:
        return x[0] + calc(x[1:])

l = [1, 2, 3, 4]
print calc(l)
```

在每一层 这个函数都递归的调用自己来计算列表剩余值的和 这个和随后的加到前面的一项中 当列表变为空的时候 递归循环结束 并返回 0 在每个层级上 要加和的列表变得越来越小 直到它变为空 变为空时循环结束

2.处理任意结构

一个简单的例子 计算一个嵌套的子列表结构中所有数字的总和

```
x = [1, [2, 3, 4], [5], [1, 2, 3, 4]]
```

```
def sumtree(x):
    res = 0
    for j in x:
        if isinstance(j, int):
            res += j
        if isinstance(j, list):
            res += sumtree(j)
    return res
print sumtree(x)
```

Python 函数的灵活性体现

1. 可以把函数对象赋值给其他名称并且引用

```
def echo(message):
    print   message
x = echo
x('hello')
```

2.可以把函数作为参数传递给另一个函数

```
def f1(func, args):
    func(args)
```

```
f1(echo, 'hello worl')
```

3.可以把函数和参数放入元组中进行循环

```
x = [(echo, 'hello'), (echo, 'world')]

for (func, args) in x:

    func(args)
```

匿名函数

lambda 函数是一种快速定义单行的最小函数

1.lambda 可以省去函数的定义过程让代码更精简

2.对于一些抽象的 不会在别的地方重复使用的函数 有时候给函数起个名字都是个难题 此时使用 lambda 不需要考虑的命名的问题

3.使用 lambda 在某些时候可以让代码更容易理解

4.lambda 是一个表达式而不是一个语句

lambda 的一般形式

lambda args1, args2: 表达式

代码示例

```
def f(x, y):
```

```
    return x * y

print f(2, 3)


g = lambda x, y: x*y

print g(3, 4)
```

在 lambda 中 冒号前是参数 可以有多个 用逗号隔开 冒号右边是返回值 构造的是一个函数的对象

为什么使用 lambda

lambda 起到了一个函数速写的作用， lambda 函数都可以用 def 函数来替代 但是在特定环境下 lambda 函数可以让代码更加的简洁

map()

```
def f1(x, y):
    return x + y
```

```
print f1(1, 2)

print f1(3, 4)



l = [1, 2, 3, 4, 5]







x = [j + 10 for j in l]

print x



print '**' * 10




res = []

for j in l:

    t = j + 10

    res.append(t)

else:

    print res



print '**' * 10
```

```
def f1(x):
    return x + 10


res1 = []
for j in l:
    t = f1(j)
    res1.append(t)
else:
    print res1



print '**' * 10



def f1(x):
    return x + 10
x   = map(f1, l)
print x


print '**' * 10
```

```
x =   map((lambda x: x + 10),l)

print x




print '**' * 10




x = map(None, [1, 2, 3, 4, 5], [3,4,5,6,7], [4, 5, 6, 7, 8])

print x




print '**' * 10




x = map((lambda x, y, z: x + y + z), [1, 2, 3, 4, 5], [3,4,5,6,7], [4, 5, 6, 7, 8])


print x
```

函数式编程

filter

filter(function, sequence)：对系列中中的每一个元素依次执行 function(item)，将执行结果为 True 的 item 组成一个 List/String/Tuple（取决于 sequence 的类型）返回：

```
>>> def f(x):
        return x % 2 != 0 and x % 3 != 0
>>> filter(f, range(2, 25))
```

reduce

python 中的 reduce 内建函数是一个二元操作函数，他用来将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给 reduce 中的函数 func()（必须是一个二元操作函数）先对集合中的第 1，2 个数据进行操作，得到的结果再与第三个数据用 func()函数运算，最后得到一个结果。

如：

```
def myadd(x,y):
    return x+y
sum=reduce(myadd,(1,2,3,4,5,6,7))
print sum
```

```
In [18]: v
Out[18]: [[1, 2, 3], [2, 3, 4], [3, 4, 5]]
In [21]: reduce(lambda x, y: [x[i]+y[i] for i in range(len(x))], v)
Out[21]: [6, 9, 12]
```