第 6 章 抽象

1.懒惰即美德

当然懒惰驱动人类的创新。

Fib 数列：

```
Fibs=[0,1]
for i in range(8):
    Fibs.append(fibs[-2] + fibs[-1])
print Fibs
```

改进：输入计算长度

```
Fibs=[0,1]
num=input(“input：”)  #使用 raw_input 更安全
 for i in range(num-2):
    Fibs.append(fibs[-2] + fibs[-1])
 print Fibs
```

抽象之后：

```
num=input(“input：”)
 print fibs(num)  #调用 fibs 函数
```

2.抽象和结构

3.创建函数

callable 内建函数 用来判断函数是否可调用

```
import math
 x=math.sqrt
 callable(x)   #python3 是 hasattr(func,__call__)
```

fibs 函数

```
def fibs(num):
   result=[0,1]
   for i in range(num-2):
       result.append(result[-2]+result[-1])
 return result
```

1.记录函数

文档字符串： 在函数开头写下字符串，它会视为函数的一部分被存储

```
def sqrt():
   ‘sqrt func’
```

访问文档字符串： sqrt.__doc__

2.关于无返回值的函数

```
def f():
  print ‘over’
```

所以的函数都返回了东西，没有返回值的都返回了 None。对于 if 等语句都要采用防御式编程来处理。

4.参数魔法

1.值从哪里来

2.我能改变参数吗？ 参数存储在局部作用域内

字符串，数字，元组不能修改，所以只能用新值覆盖。但是可以变的列表能改变么

```
def change(n):
    n[0]='Jimmy'

name=['a','b']
change(name)
name    ['jimmy','b']
```

其实这里引用了一个列表。要改变这种清楚，必须复制一个参数的副本

```
change(name[:])
```

3.关键字参数和默认值

回避位置参数,可以使用关键字参数

```
def sum(a,b):
  return a+b
sum(a=1,b=2) sum(b=2,a=1)
```

默认值：

```
def sum(a=0,b=0)
    return a+b
```

你可以不提供，提供一些，提供全部参数的值

4.收集参数

提供任意多的参数。

```
def print_par(*par):
  print par
print_par(1,2,3,4)
```

(1,2,3,4) 参数前*号将所有值放置在一个元组中

处理关键字参数：

```
def print_par(**par):
  print par
print_par(x=1,y=2,z=3) {'x':1,'y':2,'z':3}
```

**号把关键字参数放入字典

5.反转过程

使用*和**号可以反转。例子：

```
def add(x,y):
  return x+y
par=(1,2)
add(*par)
```

3 *和**只能传递元组或者字典

5.作用域

变量其实就是值的名字，当执行赋值之后如 x=1，名称 x 就引用到值 1，像字典一样，键值对。这个字典是不可见的，但是我们可以返回这种不可见的字典。函数 vars 就是做这样的事。

```
x=1
scope=vars()  #一般来说返回的字典不可修改，但是官方文档说的是未定义
scope['x']  1
```

这种不可见的字典其实就叫作用域或者命名空间。除了全局作用域以为，每个函数调用都会创建一个新的作用域。

关于全局变量
如果局部变量屏蔽了全局变量，要访问全局变量就用 **globals** 函数。
如： **def print_par(par):**
print par+globals()['par']
在函数里面声明变量会自动成为局部变量，如果要声明全局变量，就应该像这样：

```
def change(par):
   global x
    x= x+1
```

函数嵌套和闭包

```
def multiplier(factor):
   def multiplyByFactor(number):
      return number*factor
 return multiplyByFactor   #外层函数返回里层函数
```

每次调用外层函数时，它内部的函数都会被重新绑定，**fatcor** 变量有新值。

```
 double=multiplier(2)
 double(5)  10
```

6.递归
递归就不用多说了。书上举了两个例子。
1.阶乘和幂

```
  def fact(n):
     if n== 1:
        return 1
  else:
      return  n*fact(n-1)

  def power(x,n):
      if n==0:
         return  1
       for i in range(n):
           result *= x
   return result
```

2.二分查找

```
   def search(seq,number,lower,upper):
     if lower == upper:
         assert number==seq[upper]
         return upper
     else:
         middle = (lower+upper)//2
```

```
if number > seq[middle]:
    return search(seq,number,middle+1,upper)
else:
     return search(seq,number,lower.middle)
```

其实这里的代码还可以很好的优化。

lambda 表达式：

匿名函数 创建短小的函数

```
lambda x: x+1
```

python 关于函数式编程的函数还有：map filter reduce

表6-1 本章的新函数

函 数	描 述
map(func, seq [, seq, ...])	对序列中的每个元素应用函数
filter(func, seq)	返回其函数为真的元素的列表
reduce(func, seq [, initial])	等同于func(func(func(seq[0], seq[1]), seq[2]), ...)
sum(seq)	返回seq中所有元素的和
apply(func[, args[, kwargs]])	调用函数，可以提供参数