《Python统计计算》

（秋季学期）

翟祥
北京林业大学

第5章　函数的设计和使用

- 将可能需要反复执行的代码封装为函数，并在需要该段代码功能的地方调用，不仅可以实现代码的复用，更重要的是可以保证代码的一致性，只需要修改该函数代码则所有调用均受到影响。

5.1 函数定义

```
def 函数名([参数列表]):
    '''注释'''
    函数体
```

5.1 函数定义

- 斐波那契数列

```
def fib(n):
    '''定义函数，求小于n的斐波那契数列'''
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

#调用函数
fib(1000)
```

5.1 函数定义

- 在定义函数时，开头部分的注释并不是必需的，但是如果为函数的定义加上这段注释的话，可以为用户提供友好的提示和使用帮助。例如，把上面生成斐波那契数列的函数定义修改为下面的形式，加上一段注释。

```
>>> def fib(n):
        '''accept an integer n.
        return the numbers less than n in Fibonacci sequence.'''
        a, b = 1, 1
        while a < n:
            print(a, end=' ')
            a, b = b, a+b
        print()
```

5.1 函数定义

```
>>> def fib(n):
        '''accept an integer n.
           return the numbers less than n in Fibonacci sequence.'''
        a, b = 1, 1
        while a < n:
            print(a, end=' ')
            a, b = b, a+b
        print()


>>> fib(
        (n)
        accept an integer n.
        return the numbers less than n in Fibonacci sequence.
```

5.2 形参与实参

- 函数定义时括弧内为形参，一个函数可以没有形参，但是括弧必须要有，表示该函数不接受参数。
- 函数调用时向其传递实参，将实参的值或引用传递给形参。
- 在函数内直接修改形参的值不影响实参。

5.2 形参与实参

- 例1：编写函数，接受两个整数，并输出其中最大数。

```
def printMax(a, b):
    if a>b:
        pirnt(a, 'is the max')
    else:
        print(b, 'is the max')
```

- 这个程序并不是很好，如果输入的参数不支持比较运算，会出错。

5.2 形参与实参

- 对于绝大多数情况下（参数为不可变对象），在函数内部直接修改形参的值不会影响实参。例如下面的示例：

```
>>> def addOne(a):
        print(a)
        a += 1
        print(a)
>>> a = 3
>>> addOne(a)
3
4
>>> a
3
```

5.2 形参与实参

- 在有些情况下(参数为可变对象)，可以通过特殊的方式在函数内部修改实参的值，例如下面的代码。

```
>>> def modify(v): #修改列表元素值
        v[0] = v[0]+1
>>> a = [2]
>>> modify(a)
>>> a
[3]
>>> def modify(v, item): #为列表增加元素
        v.append(item)
>>> a = [2]
>>> modify(a,3)
>>> a
[2, 3]
>>> def modify(d): #修改字典元素值或为字典增加元素
        d['age'] = 38
>>> a = {'name':'Dong', 'age':37, 'sex':'Male'}
>>> a
{'age': 37, 'name': 'Dong', 'sex': 'Male'}
>>> modify(a)
>>> a
{'age': 38, 'name': 'Dong', 'sex': 'Male'}
```

5.3 参数类型

- 在Python中，函数参数有很多种：可以为普通参数、默认值参数、关键参数、可变长度参数等等。
- Python函数的定义非常灵活，在定义函数时不需要指定参数的类型，也不需要指定函数的类型，完全由调用者决定，类似于重载和泛型；
- 函数编写如果有问题，只有在调用时才能被发现，传递某些参数时执行正确，而传递另一些类型的参数时则出现错误。

5.3.1 默认值参数

- def 函数名(形参名=默认值，.....)

  函数体

- 默认值参数必须出现在函数参数列表的最右端，且任何一个默认值参数右边不能有非默认值参数。

```
>>> def f(a=3,b,c=5):
        print a,b,c

SyntaxError: non-default argument follows default argument
>>> def f(a=3,b):
        print a,b

SyntaxError: non-default argument follows default argument
>>> def f(a,b,c=5):
        print a,b,c


>>>
```

5.3.1 默认值参数

- 调用带有默认值参数的函数时，可以不对默认值参数进行赋值，也可以赋值，具有较大的灵活性。

```
>>> def say( message, times =1 ):
        print(message * times)
>>> say('hello')
hello
>>> say('hello',3)
hellohellohello
>>> say('hi ',7)
hi hi hi hi hi hi hi
```

5.3.1 默认值参数

- 再例如，下面的函数使用指定分隔符将列表中所有字符串元素连接成一个字符串。

```
>>> def Join(List,sep=None):
        return (sep or ' ').join(List)
>>> aList = ['a', 'b', 'c']
>>> Join(aList)
'a b c'
>>> Join(aList, ',')
'a,b,c'
```

5.3.1 默认值参数

- 默认值参数如果使用不当，会导致很难发现的逻辑错误，例如：

```
def demo(newitem,old_list=[]):
    old_list.append(newitem)
    return old_list
>>> print(demo('5',[1,2,3,4]))
[1, 2, 3, 4, '5']
>>> print(demo('aaa',['a','b']))
['a', 'b', 'aaa']
>>> print(demo('a'))
['a']
>>> print(demo('b'))
['a', 'b']
>>>
```

5.3.1 默认值参数

- 改成下面的样子就不会有问题了：

```
def demo(newitem,old_list=None):
    if old_list is None:
        old_list=[]
    old_list.append(newitem)
    return old_list
>>> print(demo('5',[1,2,3,4]))
[1, 2, 3, 4, '5']
>>> print(demo('aaa',['a','b']))
['a', 'b', 'aaa']
>>> print(demo('a'))
['a']
>>> print(demo('b') )
['b']
>>>
```

5.3.2 关键参数

- 关键参数是指实参，即调用函数时的参数传递方式。
- 通过关键参数传递，实参顺序可以和形参顺序不一致，但不影响传递结果，避免了用户需要牢记参数位置顺序的麻烦。

```
>>> def demo(a,b,c=5):
        print(a,b,c)
>>> demo(3,7)
3 7 5
>>> demo(a=7,b=3,c=6)
7 3 6
>>> demo(c=8,a=9,b=0)
9 0 8
```

5.3.3 可变长度参数

- 可变长度参数主要有两种形式：
  - *parameter 用来接受多个实参并将其放在一个元组中
  - **parameter 接受字典形式的实参。

5.3.3 可变长度参数

```
>>> def demo(*p):
        print(p)
>>> demo(1,2,3)
(1, 2, 3)
>>> demo(1,2)
(1, 2)
>>> demo(1,2,3,4,5,6,7)
(1, 2, 3, 4, 5, 6, 7)
```

5.3.3 可变长度参数

```
>>> def demo(**p):
        for item in p.items():
            print(item)
>>> demo(x=1,y=2,z=3)
('y', 2)
('x', 1)
('z', 3)
```

5.3.3 可变长度参数

- 几种不同类型的参数可以混合使用，但是不建议这样做

```
>>> def func_4(a,b,c=4,*aa,**bb):
        print(a,b,c)
        print(aa)
        print(bb)
>>> func_4(1,2,3,4,5,6,7,8,9,xx='1',yy='2',zz=3)
(1, 2, 3)
(4, 5, 6, 7, 8, 9)
{'yy': '2', 'xx': '1', 'zz': 3}
>>> func_4(1,2,3,4,5,6,7,xx='1',yy='2',zz=3)
(1, 2, 3)
(4, 5, 6, 7)
{'yy': '2', 'xx': '1', 'zz': 3}
```

5.3.4 参数传递的序列解包

- 传递参数时，可以通过在实参序列前加星号将其解包，然后传递给多个单变量形参。

```
>>> def demo(a,b,c):
        print(a+b+c)
>>> seq=[1,2,3]
>>> demo(*seq)
6
```

5.4 return语句

- return语句用来从一个函数中返回，即跳出函数，也可用return语句从函数中返回一个值。
- 如果函数没有return语句，Python将认为该函数以return None结束。

```
def maximum( x, y ):
    if x>y:
        return x
    else:
        return y
```

- 在调用内置数据类型的方法时，一定要注意该方法有没有返回值。

5.5 变量作用域

- 变量起作用的范围称为变量的作用域。
- 一个变量在函数外部定义和在函数内部定义，其作用域是不同的。
- 局部变量的引用比全局变量速度快。

5.5.1 局部变量

在函数内定义的变量只在该函数内起作用，称为局部变量。

函数结束时，其局部变量被自动删除。

5.5.2 全局变量

- 如果想要在函数内部给一个定义在函数外的变量赋值，那么这个变量就不能是局部的，其作用域必须为全局的，能够同时作用于函数内外，称为全局变量，可以通过global来定义。
  - （1）一个变量已在函数外定义，如果在函数内需要为这个变量赋值，并要将这个赋值结果反映到函数外，可以在函数内用global声明这个变量，将其定义为全局变量。
  - （2）在函数内部直接将一个变量声明为全局变量，在函数外没有声明，在调用这个函数之后，将增加为新的全局变量。

5.5.3 案例

```
>>> def demo():
        global x
        x = 3
        y =4
        print(x,y)
>>> x = 5
>>> demo()
3  4
>>> x
3
>>> y
出错
NameError: name 'y' is not defined
>>> del x
>>> x
出错
NameError: name 'x' is not defined
>>> demo()
3  4
>>> x
3
>>> y
出错
NameError: name 'y' is not defined
```

5.6 lambda表达式

- lambda表达式可以用来声明匿名函数，即没有函数名字的临时使用的小函数，只可以包含一个表达式，且该表达式的计算结果为函数的返回值，不允许包含其他复杂的语句，但在表达式中可以调用其他函数。

5.6 lambda表达式

```
>>> f=lambda x,y,z:x+y+z
>>> f(1,2,3)
6
>>> g=lambda x,y=2,z=3:x+y+z
>>> g(1)
6
>>> g(2,z=4,y=5)
11
>>> L=[(lambda x:x**2),(lambda x:x**3),(lambda x:x**4)]
>>> print(L[0](2),L[1](2),L[2](2))
4 8 16
>>> D={'f1':(lambda:2+3),'f2':(lambda:2*3),'f3':(lambda:2**3)}
>>> print(D['f1'](),D['f2'](),D['f3']())
5 6 8
>>> L=[1,2,3,4,5]
>>> print(list(map((lambda x:x+10),L)))
[11, 12, 13, 14, 15]
>>> L
[1, 2, 3, 4, 5]
```

5.6 lambda表达式

```
>>> def demo(n):
    return n*n
>>> demo(5)
25
>>> a_list=[1,2,3,4,5]
>>> list(map(lambda x:demo(x),a_list))
[1, 4, 9, 16, 25]
```

5.6 lambda表达式

```
>>> data = list(range(20))
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> import random
>>> random.shuffle(data)
>>> data
[4, 3, 11, 13, 12, 15, 9, 2, 10, 6, 19, 18, 14, 8, 0, 7, 5, 17, 1, 16]
>>> data.sort(key=lambda x:x)
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> data.sort(key=lambda x:len(str(x)))
>>> data
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> data.sort(key=lambda x:len(str(x)),reverse=True)
>>> data
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

5.7 案例精选

- 例1：编写函数计算圆的面积。

```
from math import pi as PI
import types
def CircleArea(r):
    if isinstance(r,int) or isinstance(r,float): #确保接收的参数为数值
        return PI*r*r
    else:
        print('You must give me an integer or float as radius.')

#调用函数
print(CircleArea(3))
```

5.7 案例精选

- 例2：编写函数，接收任意多个实数，返回一个元组，其中第一个元素为所有参数的平均值，其他元素为所有参数中大于平均值的实数。

```
def demo(*para):
    avg = sum(para)/len(para) #注意Python 2.x与Python 3.x对除法运算符“/”的解释不同
    g = [i for i in para if i>avg]
    return (avg,)+tuple(g)

#调用函数
print(demo(1,2,3,4))
```

5.7 案例精选

- 例3：编写函数，接收字符串参数，返回一个元组，其中第一个元素为大写字母个数，第二个元素为小写字母个数。

```
def demo(s):
    result = [0,0]
    for ch in s:
        if 'a'<=ch<='z':
            result[1] += 1
        elif 'A'<=ch<='Z':
            result[0] += 1
    return result

#调用函数
print(demo('aaaabbbbC'))
```

5.7 案例精选

- 例4：编写函数，接收包含20个整数的列表lst和一个整数k作为参数，返回新列表。处理规则为：将列表lst中下标k之前的元素逆序，下标k之后的元素逆序，然后将整个列表lst中的所有元素再逆序。

```
def demo(lst,k):
    x = lst[:k]
    x.reverse()
    y = lst[k:]
    y.reverse()
    r = x+y
    r.reverse()
    return r

lst = list(range(1,21))
print(lst)
print(demo(lst,5))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1, 2, 3, 4, 5]
```

5.7 案例精选

- 本例的执行结果实际上是把列表中所有元素循环左移k位。在collections标准库的deque对象已经实现了该功能，直接调用即可。

```
>>> import collections
>>> x = range(20)
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> x = collections.deque(x)
>>> x
deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
>>> x.rotate(-3)
>>> x
deque([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2])
>>> x = list(x)
>>> x
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2]
```

5.7 案例精选

- 例5：编写函数，接收整数参数t，返回斐波那契数列中大于t的第一个数。

```
def demo(t):
    a, b = 0, 1
    while b<t:
        a, b = b, a+b
    else:
        return b

#调用函数
print(demo(50))
```

5.7 案例精选

- 例6：编写函数，接收一个包含若干整数的列表参数lst，返回一个元组，其中第一个元素为列表lst中的最小值，其余元素为最小值在列表lst中的下标。

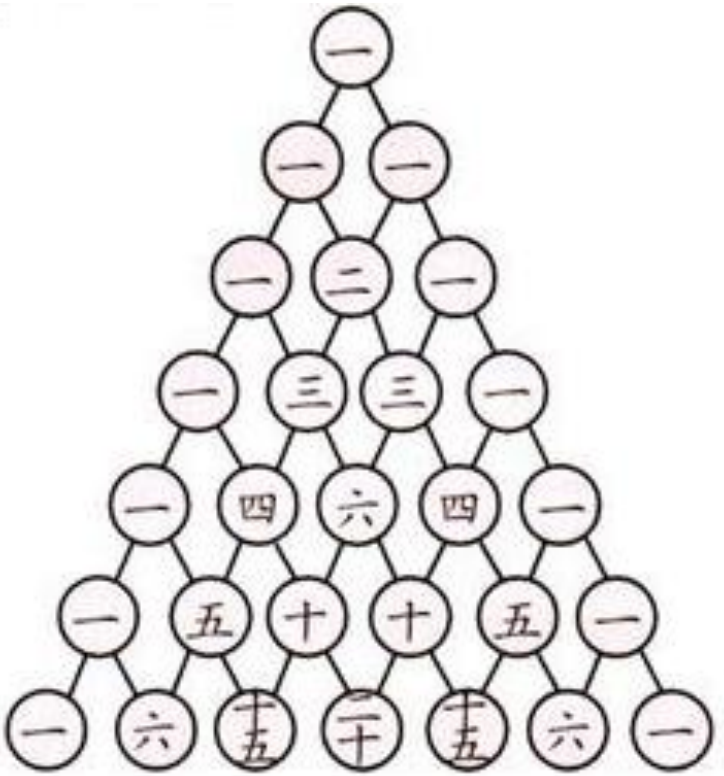
```
import random
def demo(lst):
    m = min(lst)
    result = (m,)
    for index, value in enumerate(lst):
        if value==m:
            result = result+(index,)
    return result

#主程序
x = [random.randint(1,20) for i in range(50)]
print(x)
print(demo(x))
```

5.7 案例精选

- 例7：编写函数，接收一个整数t为参数，打印杨辉三角前t行。

```
def demo(t):
    print([1])
    print([1,1])
    line = [1,1]
    for i in range(2,t):
        r = []
        for j in range(0,len(line)-1):
            r.append(line[j]+line[j+1])
        line = [1]+r+[1]
        print(line)
#调用函数
demo(10)
```



5.7 案例精选

- 例8：编写函数，接收一个正偶数为参数，输出两个素数，并且这两个素数之和等于原来的正偶数。如果存在多组符合条件的素数，则全部输出。

```
import math
def IsPrime(n):
    m = int(math.sqrt(n))+1
    for i in range(2, m):
        if n%i==0:
            return False
    return True
def demo(n):
    if isinstance(n,int) and n>0 and n%2==0:
        for i in range(3, int(n/2)+1):
            if IsPrime(i) and IsPrime(n-i):
                print(i, '+', n-i, '=', n)
demo(60)
```

5.7 案例精选

- 例9：编写函数，接收两个正整数作为参数，返回一个数组，其中第一个元素为最大公约数，第二个元素为最小公倍数。

```
def demo(m,n):
    if m>n:
        m, n = n, m
    p = m*n
    while m!=0:
        r = n%m
        n = m
        m = r
    return (int(p/n),n)
print(demo(20,30))
>>> fractions.gcd(36,39)
3
>>> fractions.gcd(20,30)
10
>>> 30*20/fractions.gcd(20,30)
60
```

5.8 高级话题

- 内置函数map可以将一个函数作用到一个序列或迭代器对象上。

```
>>> list(map(str,range(5)))
['0', '1', '2', '3', '4']
>>> def add5(v):
        return v+5
>>> list(map(add5,range(10)))
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

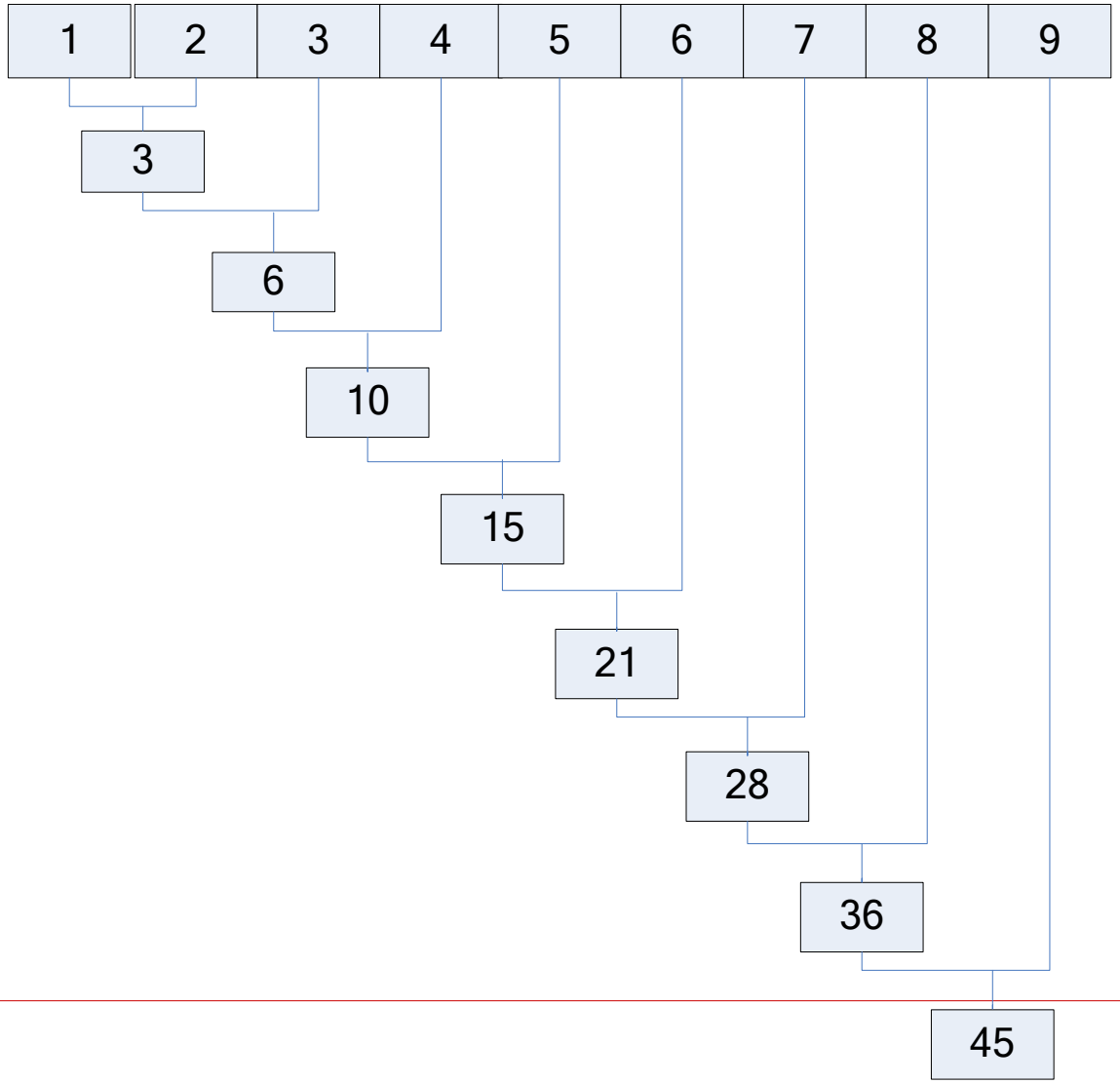
5.8 高级话题

- 内置函数reduce可以将一个接受2个参数的函数以累积的方式从左到右依次作用到一个序列或迭代器对象的所有元素上。

```
>>> seq=[1,2,3,4,5,6,7,8,9]
>>> reduce(lambda x,y:x+y, seq)
45
>>> def add(x, y):
    return x + y
>>> reduce(add,range(10))
45
>>> reduce(add,map(str,range(10)))
'0123456789'
```

在python 3中，使用reduce函数需要
from functools import reduce

5.8 高级话题



5.8 高级话题

- 内置函数filter将一个函数作用到一个序列上，返回该序列中使得该函数返回值为True的那些元素组成的列表、元组或字符串。

```
>>> seq=['foo','x41','?!','***']
>>> def func(x):
        return x.isalnum()
>>> filter(func,seq)
['foo', 'x41']
>>> seq
['foo', 'x41', '?!', '***']
>>> [x for x in seq if x.isalnum()]
['foo', 'x41']
>>> filter(lambda x:x.isalnum(),seq)
['foo', 'x41']
```

5.8 高级话题

Python中的yield

- 介绍yield前有必要先说明下Python中的迭代器(iterator)和生成器(constructor)
- 迭代器是一个实现了迭代器协议的对象，Python中的迭代器协议就是有next方法的对象会前进到下一结果，而在结果的末尾则会引发StopIteration。任何这类的对象在Python中都可以用for循环或其他遍历工具迭代，迭代工具内部会在每次迭代时调用next方法，并且捕捉StopIteration异常来确定何时离开。
- 使用迭代器的好处是：每次只从对象中读取一条数据，不会造成内存的过大开销。

比如要逐行读取一个文件的内容，利用readlines()方法，我们可以这么写：

```
for line in open("test.txt").readlines():
    print line
```

这样虽然可以工作，但不是最好的方法。因为它实际上是把文件一次加载到内存中，放在列表中，然后逐行打印。当文件很大时，这个方法的内存开销就很大了。

利用file的迭代器，我们可以这样写：

```
for line in open("test.txt"):   #use file iterators
    print line
```

这是最简单也是运行速度最快的写法，他并没显式的读取文件，而是利用迭代器每次读取下一行。

生成器函数在Python中与迭代器协议的概念联系在一起。简而言之，包含yield语句的函数会被编译成生成器。当函数被调用时，他们返回一个生成器对象，这个对象支持迭代器接口。

不像一般的函数会生成值后退出，生成器函数在生成值后会自动挂起并暂停他们的执行和状态，他的本地变量将保存状态信息，这些信息在函数恢复时将再度有效

```
>>> def g(n):
...         for i in range(n):
...             yield i **2
...
>>> for i in g(5):
...     print i,":",
...
0 : 1 : 4 : 9 : 16 :
```

要了解他的运行原理，我们来用next方法看看：

```
>>> t = g(5)
>>> t.next()
0
>>> t.next()
1
>>> t.next()
4
>>> t.next()
9
>>> t.next()
16
>>> t.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

在运行完5次next之后，生成器抛出了一个StopIteration异常，迭代终止。

再来看一个yield的例子，用生成器生成一个Fibonacci数列：

```
def fab(max):
    a,b = 0,1
    while a < max:
        yield a
        a, b = b, a+b

>>> for i in fab(20):
...     print i,",",
...
0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 ,
```

5.8 高级话题

- 生成器：惰性求值，可迭代，适用于大数据处理

```
>>> def f():
        a, b = 1, 1
        while True:
            yield a
            a, b = b, a+b
>>> a = f()
>>> a.next()
1
>>> a.next()
1
>>> a.next()
2
>>> a.next()
3
>>> a.next()
5
>>> a.next()
8
>>> a.next()
13
>>> a.next()
21
```

```
>>> for i in f():
        print i,
        if i >1000:
            break
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

```
>>> a = f()
>>> for i in range(10):
        print a.next(),
```

```
1 1 2 3 5 8 13 21 34 55
>>> for i in range(10):
        print a.next(),
```

```
89 144 233 377 610 987 1597 2584 4181 6765
```

5.8 高级话题

- 使用dis模块可以查看函数的字节码指令

```
>>> def add(n):
    n+=1
    return n
>>> import dis
>>> dis.dis(add)
  2           0 LOAD_FAST                0 (n)
              3 LOAD_CONST               1 (1)
              6 INPLACE_ADD
              7 STORE_FAST               0 (n)

  3          10 LOAD_FAST                0 (n)
             13 RETURN_VALUE
```

5.8 高级话题

函数嵌套定义

在Python中，函数是可以嵌套定义的。例如，下面的代码演示了函数嵌套定义的情况

```
def linear(a, b):
    def result(x):
        return a * x + b
return result
```

5.9 作业

- 编写函数，接收一个所有元素值都不相等的整数列表x和一个整数n，要求将值为n的元素作为支点，将列表中所有值小于n的元素全部放到n的前面，所有值大于n的元素放到n的后面。

python™

谢谢Q/A