

第四章 函数与模块

本章导读

一般说来，程序设计语言中的函数可简单地理解为用名称代表一段代码。该段代码可以被重复调用，以实现一定功能。也可以将函数理解为是程序设计任务的分解，将一个大的任务目标分解为功能不同的多个函数，以达到分而治之之目的。从这个意义上讲，函数任务的抽象。函数有助于程序维护。如果将相同或近似功能的代码分布在程序多个位置，一旦发现问题需要改正或优化升级，将多处修改，很可能导致维护不彻底等等诸多问题。

综上所述，函数对程序设计意义重大。虽然很多语言的函数大致相似，但不同语言的函数也有自己特色，而Python的函数特点颇多。

学习目标：

1. 掌握函数的定义与调用；
2. 掌握多种类型函数参数；
3. 掌握函数与类的关系；

本章目录

- 第一节 快速了解
- 第二节 数据类型
- 第三节 参数种类
 - 1、默认参数
 - 2、变参函数
 - 3、关键字参数
 - 4、命名关键字参数
- 第四节 lambda函数
- 第五节 返回值
- 第六节 函数对象
- 第七节 参数作用域
- 第八节 参数值传递
- 第九节 函数装饰器
 - 1、快速了解
 - 2、带参装饰器
- 第十节 偏函数
- 第十一节 可迭代对象
- 第十二节 生成器
- 第十三节 小结

第一节 快速了解

例程4-1能提供不少函数的相关信息。第3-4行定义了一个名为isOdd()的函数，用于判断一个数是否奇数，其中N为参数。与C/C++、Java不同，N无需也不能说明其类型，比如：定义N为整数等。通过该定义，我们可以了解到isOdd()函数有一个参数，调用时也必须传入一个参数，如例程第6行所示。如果不传入一个参数，Python将提示错误，TypeError: isOdd() missing 1 required positional argument: 'N'，其含义是：类型错误：isOdd()缺失1个要求的位置参数：N。也就是说，如果要求有一个参数而不传入，则将提示TypeError错误。

观察第8行、第9行，会发现max()函数的参数个数在变化，这种函数成为变参函数，有关变参函数，请参考本章第四节。

例程4-1

```

第1行  #快速了解函数
第2行
第3行  def isOdd(N):#定义函数
    
```

```

第4行    return N%2!=0
第5行
第6行    print(isOdd(11))#输出： True， 函数调用
第7行
第8行    print(max(12,454,875,45))#输出： 875
第9行    print(max(12,454,875,45,98,211,985))#输出： 875
第10行
第11行    listA=[12,12,345,64,34,23]
第12行    count=0
第13行    for i in listA:
第14行        print("第",count,"个数： ",i,end="\t",sep="")#命名参数函数， 命名参数的顺序可以调整
第15行        count+=1
第16行
第17行    print(min(12,234,456))#min可以接受多个参数， 可变参数函数
第18行
第19行    def Pow(x,N=2):#默认值参数
第20行        result=1
第21行        for i in range(N):
第22行            result*=x
第23行        return result
第24行
第25行    print(Pow(3))#输出： 9， 默认为2次方
第26行    print(Pow(3,3))#输出： 27， 非二次方， 输出参数
第27行
第28行    #eof

```

例程第14行调用print()函数用于输出，回忆和观察会发现，print()函数除了参数个数可以变化外，其参数有些不同，比如end='\t'，这个参数有了一个名称为end，sep也与之类似。这种参数称之为命名参数。有关命名参数的讲解，请参考本章第五节。

第19行定义一个名为Pow的函数，该函数的功能是求x的N次幂，一共两个参数。观察会发现第二个参数N其后有“=2”其功能是在省略该参数时，默认值为2，如果第25行对Pow()函数的调用，如果不是求2次幂，则需传入参数，如第26行所示。有关默认值参数，参考本章第三节。

第二节 数据类型

例程4-2是C++对函数的定义，其功能与第一节的Pow()函数相同。通过观察会发现，x和N前面都多了int，其含义是定义x和N都为整数。调用时，也必须是整数。Pow()函数前也和Python函数不同，C++或其他静态语言，要求说明返回值的类型，而Python则没有这样的要求。在定义函数时必须说明其数据类型。Python是动态语言，和静态语言有很大的不同，定义函数时无需说明数据类型。

例程4-2

```

第1行    int Pow(int x,int N){
第2行        int rtnVal=1;
第3行        for (int i=1;i<=N ;++i )
第4行            {
第5行                rtnVal*=x;
第6行            }
第7行        return rtnVal;
第8行    }

```

Python定义函数虽然不要求数据类型，但如果传入参数类型不匹配，将导致错误，比如：Pow(3,5.2)，即求3的5.2次方，此时系统会提示错误信息。能不能预先进行数据类型判断呢？答案是肯定的，如例程4-3所示。

例程4-3

```
第1行  #数据类型判断
第2行
第3行  def Pow(x,N=2):#默认值参数
第4行      if not isinstance(N,int):
第5行          raise TypeError("Pow()函数调用格式：Pow(x,N)，其中N必须为整数!")
第6行
第7行      result=1
第8行      for i in range(N):
第9行          result*=x
第10行      return result
第11行
第12行  print(Pow(3,3.3))#如此调用将触发异常
第13行
第14行  #eof
```

如例程第4行所示，isinstance(N,int)的功能是判断N(第一个参数)的类型是否是int(第二个参数)，如果是则返回True，否则返回False。类型可以是Python内置类型，如int、float、str、list、set、dict等，也可以自己定义的类型，如UpDown等。

isinstance()函数还可以判断一个值是否属于多个类型中的一种，如：isinstance([2,5,8],(list,tuple))即判断[2,5,8]是否是list或者tuple，此处返回值为True，而isinstance({'Name':'dxf','Sex':'M'},(list,tuple))的返回值为False，因为{'Name':'dxf','Sex':'M'}为dict，不是list或者tuple中的一种。

观察例程4-4，ADD()函数能对哪些数据类型适用呢？可以发现可以是第5行的str相加、可以是第6行的int相加、可以是第7行的int与float相加、也可以是第8行的list相加，甚至也可以是第19行的UpDown相加。前提条件是这些类型支持+运算符，如果不支持，则将导致错误。

例程4-4

```
第1行  #数据类型判断
第2行  def ADD(X,Y):
第3行      return X+Y
第4行
第5行  print(ADD("a","b"))
第6行  print(ADD(12,14))
第7行  print(ADD(12,12.12))
第8行  print(ADD([1,2,3],[4,5,6]))#两个list相加
第9行
第10行  class UpDown:#定义一个分数类
第11行      def __init__(self,u,d):
第12行          self.Up=u
第13行          self.Down=d
第14行      def __add__(self,argTwo):#对加号运算符定义
第15行          return UpDown(self.Up*argTwo.Down+self.Down*argTwo.Up,self.Down*argTwo.Down)
第16行      def printUD(self):
第17行          print(self.Up,"/",self.Down,sep="")
第18行
第19行  myUD=ADD(UpDown(1,2),UpDown(2,3))
```

```
第20行 myUD.printUD()#输出: 7/6
第21行
第22行 #eof
```

如何获得一个值的类型呢? Python提供的type()函数, 如例程4-5所示。

例程4-5

```
第1行 #数据类型判断
第2行
第3行 print(type(12))#输出: <class 'int'>
第4行 print(type("12"))#输出: <class 'str'>
第5行 print(type([1,2,3]))#输出: <class 'list'>
第6行 print(type((1,2,3)))#输出: <class 'tuple'>
第7行 print(type({'x':1,'y':2}))#输出: <class 'dict'>
第8行
第9行 #eof
```

第三节 参数种类

Python支持多种形式的参数, 包括: 位置参数、默认参数、变参参数、关键字参数以及命名关键字参数等。

1、默认参数

对默认关键字参数, 前面已经有所了解, 如例程4-6所示, 第3行代码POW(x,N=2)中的参数N即为默认参数, 其默认值为2。当省略参数N时, 则将N当做2处理, 如第9行所示, 如果N不为2, 则需要输入参数。

例程4-6

```
第1行 #! /usr/bin/python
第2行
第3行 def POW(x,N=2):
第4行     result=1
第5行     for i in range(0,N):
第6行         result*=x
第7行     return result
第8行
第9行 print(POW(3))#output:9
第10行 print(POW(3,3))#output:27
第11行 print(POW(3,0))#output:1
第12行
第13行 #eof
```

2、变参函数

如同Python内置函数max()既可以对两个数求最大值, 也可以对多个数求最大值, 这就说明max()支持一个或多个参数。例程4-7第3行定义了一个Python的变参函数名为Sum(), 注意参数前必须星号。第11-13行是Sum()函数的使用, 可以发现参数个数可以是0, 也可以是其他数量。

例程4-7

```
第1行 #变参函数
第2行
第3行 def Sum(*numbers):#注意:星号不能省略
第4行     #下行仅为说明问题, 一般函数定义中, 需删除本行
第5行     print(type(numbers))#输出值为<class 'tuple'>
```

```

第6行    rtnVal = 0
第7行    for i in numbers:
第8行        rtnVal +=i
第9行    return rtnVal
第10行
第11行    print(Sum(10))#1个参数，输出为10
第12行    print(Sum(10,10,20,1))#4个参数，输出为41
第13行    print(Sum())#0个参数，输出为0
第14行
第15行    #eof

```

观察例程第5行，会发现参数的类型为tuple，这就是Python变参函数的奥秘，即Python解释器将参数封装成一个tuple，然后通过for...in循环遍历每个成员，并对成员进行相应的处理。

如果要对一个list中的数求和怎么办呢？如例程4-8第14行所示，即在list前加上星号(*)，这样就可以list或者tuple的成员值当变参参数传入到函数之中。

例程4-8

```

第1行    #! /usr/bin/python
第2行
第3行    def Sum(*Nums):
第4行        valSum=0
第5行        for i in Nums:
第6行            valSum+=i
第7行
第8行        return valSum
第9行
第10行    print(Sum(1,2,3,4))#Output:10
第11行
第12行    listNum=[1,2,3]
第13行    print(Sum(listNum[0],listNum[1],listNum[2]))#Output:6
第14行    print(Sum(*listNum))#Output:6
第15行
第16行    #eof

```

3、关键字参数

关键字参数允许你传入0个或任意个含参数名称的参数，这些关键字参数在函数内部自动组装为一个dict，如例程4-9所示。

例程4-9

```

第1行    #关键字参数函数
第2行    import math
第3行    def rectTriArea(**side):#直角三角形
第4行        print(type(side))#输出<class 'dict'>
第5行        if 'a' in side and 'b' in side and 'c' in side:
第6行            if (side['c']*side['c'])!=(side['a']*side['a']+side['b']*side['b']):
第7行                raise ValueError("不构成直角三角形！")
第8行            return side['a']*side['b']/2
第9行        else:
第10行            if 'a' in side and 'b' in side:

```

```

第11行         return side['a']*side['b']/2
第12行         if 'b' in side and 'c' in side:
第13行             return math.sqrt(side['c']*side['c']-side['b']*side['b'])*side['b']/2
第14行         if 'a' in side and 'c' in side:
第15行             return math.sqrt(side['c']*side['c']-side['a']*side['a'])*side['a']/2
第16行
第17行     rectTriArea()
第18行     print(rectTriArea(a=3,b=4))
第19行     print(rectTriArea(a=3,c=5))
第20行     print(rectTriArea(c=5,b=4))
第21行     print(rectTriArea(c=5,b=4,a=3))
第22行     print(rectTriArea(c=5,b=4,a=1))
第23行
第24行     #eof

```

4、命名关键字参数

Python中常用的函数print()就有命名关键字参数的使用。如print("我的年龄",age,sep=' ',end='\t')中的sep和end就是命名关键字参数。例程4-10是关键字参数示例。和关键字参数相比，命名关键字参数只能传入指定的参数名称的参数，而关键字参数则可以传入任意参数的名称。对于例程4-9而言，传入一个参数名为d的参数会如何？

例程4-10

```

第1行     #命名关键字参数函数
第2行
第3行     def RectTri(*,a,b):
第4行         return a*b/2
第5行
第6行     print(RectTri(a=3,b=4))#输出： 6.0
第7行     print(RectTri(b=4,a=3))#输出： 6.0
第8行
第9行     #eof

```

命名关键字参数和关键字参数与位置参数不同，必须传入参数名称，否则将导致错误。另外，命名关键字参数可以有缺省值，这在有时会比较有用。

对于Python自定义函数，可以单独选用位置参数、默认参数、可变参数、关键字参数和命名关键字参数，也可以组合，不过变参参数不能和命名关键字参数混用，另外，参数定义的顺序遵从：位置参数、默认参数、可变参数/命名关键字参数和关键字参数。如例程4-11所示。

例程4-11

```

第1行     #! /usr/bin/python
第2行
第3行     def f1(a, b, c=0, *args, **kw):
第4行         print('a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw)
第5行
第6行     def f2(a, b, c=0, *, d, **kw):
第7行         print('a =', a, 'b =', b, 'c =', c, 'd =', d, 'kw =', kw)
第8行
第9行     f1(1, 2)
第10行     #output: a = 1 b = 2 c = 0 args = () kw = {}

```

```

第11行
第12行 f1(1, 2, c=3)
第13行 #output: a = 1 b = 2 c = 3 args = () kw = {}
第14行
第15行 f1(1, 2, 3, 'a', 'b')
第16行 #output: a = 1 b = 2 c = 3 args = ('a', 'b') kw = {}
第17行
第18行 f1(1, 2, 3, 'a', 'b', x=99)
第19行 #output: a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}
第20行
第21行 f2(1, 2, d=99, ext=None)
第22行 #output: a = 1 b = 2 c = 0 d = 99 kw = {'ext': None}
第23行
第24行
第25行 args = (1, 2, 3, 4)
第26行 kw = {'d': 99, 'x': '#'}
第27行 f1(*args, **kw)
第28行 #output: a = 1 b = 2 c = 3 args = (4,) kw = {'d': 99, 'x': '#'}
第29行
第30行 args = (1, 2, 3)
第31行 kw = {'d': 88, 'x': '#'}
第32行 f2(*args, **kw)
第33行 #output: a = 1 b = 2 c = 3 d = 88 kw = {'x': '#'}
第34行
第35行 #eof

```

观察例程第27、32行，会发现，对于任意函数，都可以通过类似func(*args, **kw)的形式调用它，无论它的参数是如何定义。当然，传入的参数要与定义相匹配。如删除第30、31行将发生错误，其原因是f2()的参数d是一个关键字参数，需要参数带有名称，而第25行的args含有4个值，对应到d之上，不是关键字参数。

第四节 lambda函数

在Python中，函数可以分为具名函数和匿名函数，匿名函数又称为lambda函数，例程4-12是具名函数与lambda的对比。从第5行和第6行相比，isEven()具名函数与lambda x:x%2==0的功能相同。很明显，如果一个具名函数将不会被重复使用，匿名函数更加有优势。当代码短小时，匿名函数也有更好的阅读性，且能及时就地分析其功能。当然，如果代码较长，lambda函数的优势将不存在，具名函数更有优势。在Python中，只支持一行一句。即lambda函数关键字后的分号前为输入参数，分数后为返回值，即便没有参数，分号也不能省略，如第13行所示。

例程4-12

```

第1行 #lambda函数
第2行 def isEven(N):
第3行     return N%2==0
第4行
第5行 evenListA=filter(isEven,[1,2,3,4,5,6])
第6行 evenListB=filter(lambda x:x%2==0,[1,2,3,4,5,6])
第7行 for i in evenListA:
第8行     print(i)
第9行

```

```

第10行  for i in evenListB:
第11行     print(i)
第12行
第13行  print((lambda : "x")())#输出: x
第14行  Sum2=lambda x:x+2
第15行  print(Sum2(10))#输出: 12
第16行
第17行  twoPlus=lambda x,y=5:x+y
第18行  print(twoPlus(10))#输出: 15
第19行  print(twoPlus(10,90))#输出: 100
第20行
第21行  #eof

```

Lambda函数是一种对象，可以将其赋值给一个变量，如第14、17行所示。同时，lambda函数也支持默认值，如第17、19行所示。

第五节 返回值

在Python中，函数返回值通过return关键字返回。当函数被执行时，一旦遇到return关键字，将返回函数执行结果，并跳出函数体，执行函数外的其他代码。当函数没有返回值时，函数体执行结果为None。例程4-13是关于函数返回值的示例。

例程4-13

```

第1行  #函数返回值
第2行  def addTwo(N):
第3行     return N+2
第4行  def printNum(N):
第5行     print("N=",N)
第6行
第7行  print(addTwo(10))#输出: 12
第8行  print(printNum(10))#输出: None
第9行
第10行 def pointXY(x,y):
第11行     return x,y
第12行
第13行  print(pointXY(10,20))#输出: (10,20)
第14行  print(type(pointXY(10,20)))#输出: <class 'tuple'>
第15行  print(pointXY(10,20)[0])#输出: 10
第16行
第17行  #eof

```

在Python中函数可以返回多个值，如例程第10、11行所示，同时返回x,y两个值，从第13、14行可以看出，多个返回值将被封装成Tuple类型，因此也可以通过Tuple方式访问其每个值，如第15行所示。

第六节 函数对象

在Python中，一切皆对象，函数也是对象，例程4-14是函数对象的示例。例程第3行、第6行、第9行分别定义了三个判断奇数、偶数和质数的函数，但第15行的check()函数有些不同，虽然还是传入两个参数，但fn参数在参数上也没有异常，但函数体内的fn(N)表明fn是一个函数，也就是说可以将函数作为参数传入。

例程4-14

```

第1行  #函数对象

```



```

第2行
第3行  def isOdd(N):
第4行     return N%2==1
第5行
第6行  def isEven(N):
第7行     return N%2==0
第8行
第9行  def isPrime(N):
第10行     for i in range(2,N):
第11行         if N%2==0:
第12             return False
第13行         return True
第14行
第15行  def check(N,fn):
第16行     return fn(N)
第17行
第18行  print(type(isOdd))#输出: <class 'function'>
第19行
第20行  numOE=isOdd
第21行  print(type(numOE))#输出: <class 'function'>
第22行
第23行  #函数名称成为参数?
第24行  print(check(15,isEven))#输出: False
第25行  print(check(15,isOdd))#输出: True
第26行  print(check(10,isPrime))#输出: False
第27行
第28行
第29行  #eof

```

函数参数分为：形式参数和实际参数。当定义函数时，函数的参数为形式参数简称形参；当调用函数时，函数的参数为实际参数简称实参。所谓实际参数，也就是一个具体的值，如abs(-5)中的-5是一个具体的值。

例程第24行代码check(15,isOdd)向check()函数传入函数名称，函数名称也是一个具体的值，为函数对象。函数定义如同类的定义，函数名称作为参数，如同类实例化为对象，此时称之为函数对象。

第七节 参数作用域

观察例程4-15，a和b在TestA()函数体外被分别赋值为100，在TestA()函数体内分别被赋值为200。第10、11行分别执行TestA()函数，会发现a和b的值没有发生任何变化，如同函数体内外是独立空间，互不影响。

例程4-15

```

第1行  #函数对象
第2行
第3行  a=100
第4行  b=100
第5行  def TestA(a):
第6行     a=200
第7行     b=200
第8行  print("a=",a,"b=",b)#输出: a= 200 b= 200

```

第9行	
第10行	TestA(a)
第11行	TestA(b)
第12行	
第13行	print("a=",a,"b=",b)#输出: a=100,b=100
第14行	
第15行	#eof

观察例程4-16会发现虽然TestA()函数内没有定义b, 但由于函数上一层定义有b, 函数内可以读取该值, 但是不能改变其值, 如a虽然在上层也有相同名称的变量, 但其改变并不能影响上层变量。

例程4-16

第1行	#参数作用域
第2行	
第3行	a=100
第4行	b=100
第5行	def TestA(a):
第6行	a=200
第7行	print("a=",a,"b=",b)#输出: a= 200 b= 100
第8行	
第9行	TestA(a)
第10行	TestA(b)
第11行	
第12行	print("a=",a,"b=",b)#输出: a=100,b=100
第13行	
第14行	#eof

观察例程4-17第7行, 变量b前面增加了global关键字, 从第14行的输出看, TestA()中修改b的值, 同时也修改了其上层b的值。当没有global关键字时, 只能读取但不能改变外层变量的值。另外, 在TestA()函数中, global关键字不能加在变量a的前面, 其原因是a在TestA()函数中是局部变量或者成为函数体变量, 不能使用global关键字。在本例中, a和b可以成为全局变量。例程4-18是全局变量的另外一个应用示例。

例程4-17

第1行	#参数作用域
第2行	
第3行	a=100
第4行	b=100
第5行	def TestA(a):
第6行	a=200
第7行	global b
第8行	b=200
第9行	print("a=",a,"b=",b)#输出: a= 200 b= 200
第10行	
第11行	TestA(a)
第12行	TestA(b)
第13行	
第14行	print("a=",a,"b=",b)#输出: a=100,b=200
第15行	
第16行	#eof

```

第1行  #参数作用域
第2行  numCount=0
第3行  def runIT():
第4行      global numCount
第5行      numCount+=1
第6行
第7行  runIT()
第8行  runIT()
第9行  runIT()
第10行
第11行  print("numCount=",numCount)
第12行
第13行  #eof

```

将nonlocal改成global将导致错误。

```

第1行  #参数作用域
第2行
第3行  def Test():
第4行      numCount=0
第5行      def PlusOne():
第6行          nonlocal numCount
第7行          numCount+=1
第8行          return numCount
第9行      return PlusOne#返回一个函数
第10行
第11行  myFun=Test()
第12行  print(myFun()) #输出： 1
第13行  print(myFun()) #输出： 2
第14行  print(myFun()) #输出： 3
第15行
第16行  #eof

```

从上面的例子可以看出，Python变量作用域优先顺序如下：局部变量→外层作用域变量→当前模块中的全局变量→Python内置变量，从左到右，优先顺序逐渐降低。

第八节 参数值传递

在Python中，对于可变对象，采用引用方式，即可以在函数中对值进行修改，如：list、dict等，对于不可变对象，如：tuple、str采用值传递，如例程4-20所示。

```

第1行  #参数值传递
第2行
第3行  intNum=100
第4行  boolNum=True
第5行  strNum="ABC"

```

```

第6行 listNum=[12,23,34]
第7行 tupleNum=(12,23,34)
第8行
第9行 def Test(transIn):
第10行     if isinstance(transIn,int):
第11行         print("isinstance(transIn,int)",end='\t')
第12行         transIn=999
第13行     if isinstance(transIn,bool):
第14行         print("isinstance(transIn,bool)",end='\t')
第15行         transIn=not transIn
第16行     if isinstance(transIn,str):
第17行         print("isinstance(transIn,str)",end='\t')
第18行         transIn="XYZ"
第19行     if isinstance(transIn,list):
第20行         print("isinstance(transIn,list)",end='\t')
第21行         transIn[0]=999
第22行     if isinstance(transIn,tuple):
第23行         print("isinstance(transIn,tuple)",end='\t')
第24行         #transIn[0]=999, 此处错误, Tuple不能修改
第25行         pass
第26行
第27行     return transIn
第28行
第29行 print(Test(intNum))#输出: isinstance(transIn,int) 999
第30行 print(intNum)#输出: 100
第31行
第32行 print(Test(boolNum))#输出: isinstance(transIn,int) 999
第33行 print(boolNum)#输出: True
第34行
第35行 print(Test(strNum))#输出: isinstance(transIn,str) XYZ
第36行 print(strNum)#输出: ABC
第37行
第38行 print(Test(listNum))#输出: isinstance(transIn,list) [999, 23, 34]
第39行 print(listNum)#输出: [999, 23, 34]
第40行
第41行 print(Test(tupleNum))#输出: isinstance(transIn,tuple) (12, 23, 34)
第42行 print(tupleNum)#输出: (12, 23, 34)
第43行
第44行 #eof

```

第九节 函数装饰器

1、快速了解

函数装饰是在不改变原函数功能的前提下，附加新的特征，是为函数装饰器。函数名称是一个对象或者实例或者值，这是函数装饰的基础，在本章函数对象一节已经阐明函数名称是函数对象。例程4-21定义一个名为timeIt()的函数，用于计算无参函数的耗时，SumA()和SumB()分别是两个无参函数。观察第19行代码，会发现timeIt()函数以一个函数名SumA为参数，即一个函数对象为

参数，也即是说timelt()函数的形式参数Fn的取值为SumA。在timelt()函数中，定义一个名为wrapper()的函数，并将该函数作为函数对象返回。在第19行代码中，等号右边timelt(SumA)执行完毕后，得到的返回值为wrapper，然后将wrapper函数对象赋值给sosoA。第20行代码sosoA()实际上是wapper()被执行，首先获得起点时间，然后执行SumA()函数，然后输出执行该函数的耗时。如果按第22、23行执行，则SumA()函数已经被wrapper()函数取代，执行SumA()函数将不仅得到计算结果，还将显示耗时。

例程4-21

第1行	#函数修饰器
第2行	import time
第3行	
第4行	def timelt(Fn):
第5行	print("Now! I am decorating Fn!")
第6行	def wrapper():
第7行	start=time.clock()
第8行	Fn()
第9行	end=time.clock()
第10行	print("耗时: ",end-start,end='\n\n')
第11行	return wrapper
第12行	
第13行	def SumA():#计算: 1-1千的和
第14行	Result=0
第15行	for i in range(1,1000+1):
第16行	Result+=i
第17行	print("1到1000的和: ",Result)
第18行	
第19行	sosoA=timelt(SumA)
第20行	sosoA()
第21行	
第22行	SumA=timelt(SumA)
第23行	SumA()
第24行	
第25行	@timelt
第26行	def SumB():#计算: 1-1万的和
第27行	Result=0
第28行	for i in range(1,10000+1):
第29行	Result+=i
第30行	print("1到10000的和: ",Result)
第31行	
第32行	SumB()
第33行	
第34行	#eof

观察第25-30行，会发现SumB()与SumA()结构完全一致，仅仅循环次数不同，但SumB()前面增加了@timelt，其含义是将SumB作为函数timelt()函数的参数，与SumB=timelt(SumB)等效。执行第32行SumB()与第23行SumA()模式相同。

例程4-21中被装饰的函数都没有参数，例程4-22是对带参数函数的支持，通过比较会发现例程第8行非常关键，将能支持各种参数形式，SumA()、Add()以及SumN()是其示例。

例程4-22

第1行	#函数修饰器
-----	--------

```

第2行 import time
第3行
第4行 def timelt(Fn):
第5行     print("Now! I am decorating Fn!")
第6行     def wrapper(*args,**kwargs):
第7行         start=time.clock()
第8行         Fn(*args,**kwargs)
第9行         end=time.clock()
第10行         print("耗时: ",end-start,end='\n\n')
第11行     return wrapper
第12行
第13行
第14行 @timelt
第15行 def SumA():#计算: 1-1千的和
第16行     Result=0
第17行     for i in range(1,1000+1):
第18行         Result+=i
第19行     print("1到1000的和: ",Result)
第20行
第21行 @timelt
第22行 def Add(x,y):
第23行     print(x+y)
第24行
第25行 @timelt
第26行 def SumN(*L):
第27行     Result=0
第28行     for i in L:
第29行         Result+=i
第30行     print(Result)
第31行
第32行 SumA()
第33行 Add(10,20)
第34行 SumN(10,20,30,40,50)
第35行
第36行 #eof

```

2、带参装饰器

函数装饰器可以带有参数，如例程4-23所示。

例程4-23

```

第1行 #函数修饰器
第2行 import time
第3行
第4行 def timelt(isTrace=True):
第5行     if isTrace:
第6行         def Delegator(Fn):
第7行

```

```

def wrapper(*args,**kwargs):
第8行     start=time.clock()
第9行     Fn(*args,**kwargs)
第10行    end=time.clock()
第11行    print("耗时: ",end-start,end='\n\n')
第12行    return wrapper
第13行
第14行    else:
第15行        def Delegator(Fn):
第16行            return Fn
第17行        return Delegator
第18行
第19行
第20行    @timelt(True)#等价于： SumA=timelt(True)(SumA)
第21行    def SumA():#计算： 1-1千的和
第22行        Result=0
第23行        for i in range(1,1000+1):
第24行            Result+=i
第25行        print("1到1000的和: ",Result)
第26行
第27行    @timelt(False)#等价于： Add=timelt(False)(Add)
第28行    def Add(x,y):
第29行        print(x+y)
第30行
第31行    #timelt()即便没有参数，也不能省略括号
第32行    @timelt()#等价于： SumN=timelt(True)(Add)
第33行    def SumN(*L):
第34行        Result=0
第35行        for i in L:
第36行            Result+=i
第37行        print(Result)
第38行
第39行    SumA()
第40行    Add(10,20)
第41行    SumN(10,20,30,40,50)
第42行
第43行    #eof

```

第十节 偏函数

偏函数是对有命名参数函数的封装，以方便使用，如例程4-24所示。int()函数能将多种进制的字符串转换为十进制整数，如果频繁需要进行十六进制字符串转换，可以如第7行所示，申明一个名为intHex()的自定义函数，专门用于将十六进制字符串转换为十进制整数。类似功能可以由Python模块functools协助完成，如第14行所示，更加简明方便。

例程4-24

```

第1行    #偏函数的使用
第2行

```

```

第3行 numA=int("789")#int()函数将字符串转换为整数，默认为十进制字符串
第4行 numB=int("ABC",base=16)#指定int()函数将十六进制字符串转换为整数
第5行 print(numB)
第6行
第7行 def intHex(x,base=16):#用于将十六进制字符串转换为整数
第8行     return int(x,base)
第9行
第10行 numC=intHex("FF")
第11行 print(numC)#输出: 255
第12行
第13行 import functools#装入functools模块
第14行 int16=functools.partial(int, base=16)#固定为16进制
第15行
第16行 numD=int16("FFF")#输出: 4095
第17行 print(numD)
第18行
第19行 #eof

```

另外，functools有很多与函数相关的工具，请参阅相关章节。

第十一节 可迭代对象

for-in是Python的重要循环语句，常用于list、tuple、string、range等序列以及dict、set等。for-in的使用前提必须是可迭代对象，也就是说list、tuple等都是可迭代对象，例程4-25的Odd是自定义类，由于其中定义了__iter__()和__next__()方法，其实例化对象为可迭代对象。

例程4-25

```

第1行 #!/usr/bin/python
第2行
第3行 class Odd():
第4行     def __init__(self):
第5行         self.__nowNum=1
第6行     def __iter__(self):
第7行         return self
第8行
第9行     def __next__(self):
第10行         if self.__nowNum>100:
第11行             raise StopIteration
第12行
第13行         rtnVal=self.__nowNum
第14行         self.__nowNum+=2
第15行         return rtnVal
第16行
第17行 for i in Odd():
第18行     print(i,end="\t")#输出: 1-100之间的奇数
第19行
第20行 print(Odd().__next__())#输出: 1
第21行 print(Odd().__next__())#输出: 1

```



```

第22行
第23行 myOdd=Odd()
第24行 print(myOdd.__next__())#输出: 1
第25行 print(myOdd.__next__())#输出: 3
第26行
第27行 for i in myOdd:
第28行     print(i,end="\t")#输出: 5-99之间的奇数
第29行
第30行 #print(myOdd.__next__())#输出: 跑输异常, StopIteration
第31行
第32行 iterOdd=iter(Odd())
第33行 print(type(iterOdd))#输出: <class '__main__.Odd'>
第34行
第35行 for i in iterOdd:
第36行     print(i,end="\t")#输出: 1到100之间的奇数
第37行
第38行 #eof

```

判断一个对象是否可以迭代, 可用例程4-26所示代码。

例程4-26

```

第1行 #!/usr/bin/python
第2行
第3行 from collections import Iterable
第4行 print(isinstance('123',Iterable))
第5行
第6行 #eof

```

第十二节 生成器

例程4-25的功能是生成奇数序列, 功能并不复杂, 但代码量大, Python提供了一种使用yield的替代解决方案, 其所在函数常被成为生成器, 如例程4-27所示。观察第9-10行会发现, 函数Test()能使用for-in循环, 虽然没有__next__()方法, 但也能在第13-15行使用, 且遍历到终点后, 能抛出异常StopIteration(如第16行所示, 为了能让其后续代码执行, 此处注释掉)。如同第18、19行所示, iter()函数同样能作用于Test()。值得注意: type(Test)其返回值为<class 'function'>, 而type(Test())的返回值为<class 'generator'>。

例程4-27

```

第1行 #探索yield使用
第2行
第3行 def Test():
第4行     yield 1
第5行     yield 3
第6行     yield 5
第7行
第8行 myTest=Test()
第9行 for i in myTest:
第10行     print(i,end="\t")#输出: 1 3 5
第11行
第12行

```

```

hisTest=Test()
第13行 print(hisTest.__next__())#输出: 1
第14行 print(hisTest.__next__())#输出: 3
第15行 print(hisTest.__next__())#输出: 5
第16行 #print(hisTest.__next__())#输出: StopIteration
第17行
第18行 iterTest=iter(Test())
第19行 print(next(iterTest))#输出: 1
第20行
第21行 print(type(Test))#输出: <class 'function'>
第22行 print(type(Test()))#输出: <class 'generator'>
第23行 print(type(iterTest))#输出: <class 'generator'>
第24行
第25行 #eof

```

如前所述, yield所在函数为生成器(generator)。当函数执行时, 当第1次遇到yield时, 即将yield后的表达式作为函数的返回值。第2次调用时, 从上一次yield执行后的位置开始执行, 并根据上一次的场景(如: 变量值等等)得到第2个yield后表达式的值。依次类推, 当所有yield都执行完毕后, 自动抛出StopIteration。另外, 如第21行所示, 函数名Test的类型依然是function, 而Test()则是generator(生成器), 如第22行所示。

例程4-28和例程4-29的功能都是生成质数列表, 分别采用class和yield方式, 可以明显看出, yield生成器模式更简洁。注意: 例程4-29没有raise StopIteration, 而例程4-28必须有raise StopIteration。对于生成器, yield关键字是关键。在Python语言中, 当一个yield执行完毕后, 该函数将保持当时执行环境, 包括: 局部变量、指令指针、内部堆栈、异常处理等等。当再次执行时, 将从上一个yield后开始执行。

例程4-28

```

第1行 #!/usr/bin/python
第2行 #!/usr/bin/python
第3行
第4行 class Prime():
第5行     def __init__(self,N):
第6行         self.__lastNum=N
第7行         self.__nowNum=2
第8行     def __iter__(self):
第9行         return self
第10行     def __next__(self):
第11行         for i in range(self.__nowNum,self.__lastNum+1):
第12行             for j in range(2,i//2+1):
第13行                 if i%j==0:
第14行                     break
第15行             else:
第16行                 self.__nowNum=i+1
第17行                 return i
第18行         raise StopIteration
第19行
第20行 for i in Prime(100):
第21行     print(i,end="\t")
第22行

```

对于例程4-29，for-in循环到2到N+1之间(第5行代码)，当循环完毕后，将自动执行隐含的raise StopIteration。在for-in循环体内，如果外层循环变量i(第5行代码)能被内层循环变量n(第六行代码)整除，则表明不是质数，中断内层循环，外层循环继续。如果不能被内层循环的所有数整除，则执行第8行代码，执行其中的yield i，该数由于不能被内层循环的所有数整除，因此是质数。

例程4-29

第1行 | #!/usr/bin/python

第2行 |

第3行 | def Prime(N):

第4行 | if N<2:raise StopIteration

第5行 | for i in range(2,N+1):

第6行 | for n in range(2,i//2+1):

第7行 | if i%n==0:break;

第8行 | else:yield i

第9行 |

第10行 | for i in Prime(97):

第11行 | print(i,end="\t")

第12行 |

第13行 | #eof

第十三节 小结