第5章 函数与模块

CHAPTER 5

计算机程序设计的一个重要方法是“模块”化编程方法，这种程序设计方法是将实现的功能细分为基本的功能单元，这些功能单元用“函数”实现，称为“模块”。在 Python 语言中，模块有特别的含义，Python 语言的模块专用于指包含了 Python 代码的文件。在本书前面的章节中，就已经将“文件”和“模块”两种说法统一了。

在学习了 Python 语言的数据表示和基本控制语法后，就可以编写具有任意算法功能的程序了。函数的主要作用在于通过函数可以管理冗长的代码，方便程序代码的复用、阅读、调试和交流。可以将函数在模块中的作用理解为中药店铺中小抽屉的作用。Python 语言本身具有的函数，如 print 等，称为内置函数。内置函数可以直接调用，其调用形式为“内置函数名()”或“内置函数名(参数表)”，例如，“print()”表示调用内置函数 print 输出一个空行；“print('Hello world!')”表示调用内置函数 print 输出提示信息“Hello world!”；“s=sum([1,2,3])”表示调用内置函数 sum 计算列表[1,2,3]中全部元素的和，将其赋给 s。

本章将首先介绍 Python 语言常用的内置函数。调用内置函数的方法为内置函数名加上一对圆括号，圆括号内包括向内置函数传递的实际参数(要与内置函数定义中的(形式)参数类型和数量相匹配)。然后，介绍自定义函数和递归函数的创建与调用方法，最后介绍 Python 语言中常用的模块及包。

本章的学习重点：

(1) 了解 Python 语言常用的内置函数、程序包和模块。

(2) 掌握自定义函数的设计方法。

(3) 熟练掌握递归函数的设计方法。

(4) 学会应用复合函数进行程序设计。

5.1 常用内置函数

在 Python 语言中，调用一个函数前需要先定义该函数，或者装载该函数所在的模块(或包)，也就是说，函数必须先定义后使用。调用 Python 内置函数同样遵循先定义后使用的规则，但是无须为内置函数装载包或声明，Python 语言“解释器”自动识别内置函数并完成内置函数的执行。

在 Python 官网文档中包含一个“The Python Library Reference”文档，其中“Build-in Functions”一章按字母顺序罗列了全部的内置函数，在版本 3.10.4 中共有 71 个，并详细地

介绍了每个内置函数的用法。这些函数有些极不常用，例如其中的数学函数均可被模块math中同功能的数学函数替代，模块math包含的函数请参考“The Python Library Reference”的“Numeric and Mathematical Modules”，使用math模块，需要借助“import math”装载它。

这里仅讨论最常用的53个内置函数(包括前述章节中介绍过的内置函数)，如表5-1所示。

表5-1 常用Python内置函数

序号	函数名	含义	典型用法
1	abs	求整数或浮点数的绝对值或求复数的模	abs(－4＋3j)返回5.0
2	all	全称谓词，当其参数全为真时，返回真；否则返回假	all((3,0,1))返回False；all(['a',3,7])返回True；参数为空列表或空元组时，返回真
3	any	存在谓词，当其参数全为假时，返回假；否则返回真	any((3,0,1))返回真；any参数为空列表或空元组时返回假
4	bin	求一个整数的二进制字符串	bin(－35)返回－0b100011
5	bool	返回逻辑值真或假：非零参数返回真；0或空返回假	bool('a')返回True；bool()返回False
6	chr	将整数(作为Unicode码)转化为字符	chr(65)返回“A”

7	@classmethod	声明其后的方法为类方法(使用类名调用的方法，而不是对象调用的方法)，类方法又称静态方法，对象调用的方法属于动态方法(因为对象创建后，才创建对象调用的方法)	class myMath:
			　　@classmethod
			def max(cls,a,b):
			　　return a if a＞b else b
			调用时：
			myMath.max(3,5)返回5

8	complex	返回复数	complex(3,5)返回3＋5j，也可以直接输入3＋5j(注意，只能用j)
9	delattr	删除对象的属性	delattr(obj,'x')相当于del obj.x，这里obj为对象，x为obj对象的属性(类相关内容见第6章)
10	dict	生成字典	见4.3节
11	dir	无参数时，返回局部标签列表；对象名或类名为参数时，返回对象或类的方法列表	dir(list)返回列表的内置方法
12	divmod	divmod(a,b)得到元组(a//b,a % b)	divmod(33,9)得到(3,6)
13	enumerate	enumerate(迭代器对象，start＝0)，start为枚举起始值，返回一个枚举对象	enumerate(['a','b','c','d'],start＝3)返回一个枚举对象，转化为列表为：[(3, 'a'), (4, 'b'), (5, 'c'), (6, 'd')]
14	eval	执行字符串表达式	x＝3 y＝eval('pow(x,2)') ＃得到y＝9

续表

序 号	函 数 名	含 义	典型用法
15	filter	filter(谓词函数,迭代器)返回由使“谓词函数”为真的“迭代器”中的元素组成的新迭代器	filter(lambda x: x % 2 == 0, range(1, 10+1))转化为列表为[2, 4, 6, 8, 10]
16	float	将整数或字符串转化为浮点数,这里的“字符串”只能为数字0～9,可以带有前导的正、负号	float('-5.1')返回浮点数-5.1
17	format	格式化字符串	见2.5节
18	getattr	getattr(obj, 'x')返回对象obj的属性x的值	类相关内容见第6章
19	globals	返回包含当前所有全局变量的字典	
20	hasattr	hasattr(obj, 'x')判断对象obj是否具有属性x	见第6章
21	hex	返回一个整数的十六进制字符串	hex(185)得到字符串“0xb9”
22	id	id(x)获取x的内存地址	
23	input	标准输入函数	见2.1节
24	int	将数值或字符串转化为整数	int(-3.4)返回-3 int('0xF2',16)返回242
25	isinstance	isinstance(obj,类型)判断对象obj是否为该“类型”,若是,返回True;否则,返回False	isinstance(3,int)返回True
26	issubclass	issubclass(sclass,pclass)判断sclass是否为pclass的子类,若是,返回True;否则,返回False	

27	iter	创建一个迭代器对象,具有两个参数,第2个参数为可选参数。若无第2个参数,第1个参数应为可迭代对象;若有第2个参数,第1个参数还需要具有next方法,如果遍历第1个参数遇到等于第2个参数的对象时,触发StopIteration异常(停止迭代)	class A:
			  def __init__(self,a):
			  self.x=a
			  self.idx=0
			  def __next__(self):
			  cur =self.x[self.idx]
			  self.idx+=1
			  return cur
			  def __call__(self, *args, **kwargs):
			  return self.__next__()
			if __name__=='__main__':
			  o=A([1,2,3,4,5,6,7])
			  for e in iter(o,5):
			  print(e,end=' ')
			将输出“1 2 3 4”(类的内容见第6章)

28	len	返回对象的元素个数	len((3,4,5))得到3
29	list	列表函数	见2.4节
30	locals	返回包含当前局部变量的字典	

续表

序　号	函 数 名	含　义	典 型 用 法
31	map	映射函数 map(函数,迭代器对象),将函数作用于迭代器对象的每个元素上	map(lambda x,y: x+y,[1,2,3],[4,5,6])返回结果转化为列表为[5,7,9]
32	max	返回可迭代器对象中的最大值元素	max([3,11,10,7])返回 11
33	min	返回可迭代器对象中的最小值元素	max([3,11,10,7])返回 3
34	next	next(迭代器)返回“迭代器”的当前元素,当“迭代器”遍历完后,继续调用 next 将触发 StopIteration 异常; next(迭代器, def)与“next(迭代器)”含义相同,但是当“迭代器”遍历完后,继续调用 next 将输出“def”而不触发 StopIteration 异常	it=iter([4,5,6]) print(next(it,5)) print(next(it,5)) print(next(it,5)) print(next(it,5)) 上述将输出: 4　#第一次调用 next 输出值 5　#第二次调用 next 输出值 6　#第三次调用 next 输出值 5　#迭代器已遍历完后的输出值
35	open	创建文件对象	见第 7 章
36	ord	ord(字符)返回“字符”的 Unicode 码,为函数 chr 的反作用函数	ord('a')返回 97
37	pow	pow(x,y)返回 x^y; pow(x,y,z)返回 $x^y \% z$	pow(5,2,7)得到 4
38	print	标准输出函数	见 2.1 节

39	property	用于指定类的 set 方法、get 方法、del 方法(删除属性)和 doc 方法(属性说明)
		或使用如下形式:
		class A:
		　def __init__(self):
		　　self.x=0
		　@property
		　def xp(self):
		　　return self.x
		　@xp.setter
		　def xp(self,v):
		　　self.x=v
		　@xp.deleter
		　def xp(self):
		　　del self.x
		if __name__=='__main__':
		　a=A()
		　a.xp=3
			class A:
			　def __init__(self):
			　　self.x=0
			　def getX(self):
			　　return self.x
			　def setX(self,v):
			　　self.x=v
			　def delX(self):
			　　del self.x
			　xp = property(getX, setX, delX,"x's property.")
			if __name__=='__main__':
			　a=A()
			　a.xp=3
			上述 a.xp=3 将自动执行 setX 方法,设置对象 a 的 x 为 3(好处在于保护了内部的 x),而读 a.xp 将调用 getX 方法,del a.xp 将执行 delX 方法(类的内容见第 6 章)

40	range	range(i,j,k)生成一个整数序列,从 i 至 j−1,步长为 k。i 可省略,默认为 0; k 可省略,默认为 1	list(range(5))返回[0,1,2,3,4]

续表

序　号	函 数 名	含　义	典 型 用 法
41	reversed	reversed(对象)将可迭代的对象逆序排列	list(reversed([3,5,8,2]))返回[2, 8, 5, 3]
42	round	四舍五入函数。round(数值)返回与“数值”最近的整数；round(数值,n)返回与“数值”最近的保留 n 位小数的数值	round(3.8756,2)返回 3.88
43	setattr	setattr(obj, 'x',v)将对象 obj 的属性 x 设置为 v(x 必须为对象 obj 的属性)	setattr(obj, 'x',3)等价于 obj.x=3,这里 obj 为对象。 这里承接第 39 号的表格,例如： setattr(a,'xp',5)等价于 a.xp=5
44	set	集合函数	见 4.2 节
45	slice	slice(a,b,c)等价于 a：b：c(注：从 a 按步长 c 递增到 b−1)	
46	sorted	对可迭代对象排序,具有关键字参数 reverse,reverse = True 表示降序排列；reverse=False 为默认值,表示升序排列	sorted([8,3,11,4,12,9],reverse=True)返回[12, 11, 9, 8, 4, 3]
47	@staticmethod	指示创建静态方法 注意：与@classmethod 的区别	class myMath： 　　@ staticmethod 　　def max(a,b)： 　　　　return a if a>b else b 调用时： myMath.max(3,5)返回 5
48	str	字符串函数	见 2.5 节
49	sum	sum(obj, start=0)对对象求和,关键字参数 start=0 可省略,表示从索引 0 开始累加	sum(range(0,10+1),start=3)返回 58
50	super	类的子类中使用,用于指代父类对象	类相关内容见第 6 章
51	tuple	元组函数	见 4.1 节
52	type	type(obj)返回对象 obj 的类型	
53	zip	zip(obj1,obj2,…,objn)为多个具有相同长度的对象建立相同位置的元素间的关联,结果为 zip 对象,其中的每个元素以元组形式存在	a=zip([1,3,5],[2,4,6],[13,15,17]) for e in a： 　　print(e) 上述代码得到： (1, 2, 13) (3, 4, 15) (5, 6, 17)

表 5-1 中列举的 Python 语言内置函数必须熟练掌握,有些内置函数将在第 6、7 章进一步介绍。这个表需要花大量时间反复阅读记忆和上机实验。

下面程序段 5-1 使用了表 5-1 中的几个内置函数,以实现列表的排序和求和操作,并展示 Python 内置函数可以直接调用的特点。

视频讲解

程序段 5-1　Python 内置函数直接调用演示实例

```
1        if_name_ == '__main__':
2        a = [6,4,9,10,13,2,8,5]
3        b = sorted(a)
4        print(f'Original:{a}')
5        print(f'Sorted:{b}')
6        c = reversed(b)
7        print(f'Reversed:{list(c)}')
8        print(f'Length of a:{len(a)}')
9        d = list(zip(a,b))
10       print(f'List of zip(a,b):\n{d}')
11       u = list(map(lambda x:sum(x),d))
12       print(f'Sum of each tuple:{u}')
```

在程序段 5-1 中，使用的内置函数有 list、sorted、print、format(隐式)、reversed、len、zip、map、sum 等内置函数。

第 2 行“a=[6,4,9,10,13,2,8,5]；”定义列表 a。第 3 行“b=sorted(a)”将列表 a 按升序排列。第 4 行“print(f'Original：{a}')”输出列表 a。第 5 行“print(f'Sorted：{b}')”输出排序后的列表 b。第 6 行“c=reversed(b)”将列表 b 逆序排列，返回一个迭代器对象，赋给 c。第 7 行“print(f'Reversed：{list(c)}')”输出 c。第 8 行“print(f'Length of a：{len(a)}')”输出列表 a 的长度。

第 9 行“d=list(zip(a,b))”调用 zip 函数建立列表 a 与 b 的关联，即将 a 和 b 同位置的元素组合成一个元组，并将得到的迭代器对象转化为列表，赋给 d。

第 10 行“print(f'List of zip(a,b)：\n{d}')”输出列表 d，这里“\n”表示换行符。第 11 行“u=list(map(lambda x：sum(x),d))”调用 map 函数将 lambda 函数映射到列表 d 的每个元素(这里为元组)上，计算每个元素(即元组中的所有元素)的和，计算结果组成列表 u。第 12 行“print(f'Sum of each tuple：{u}')”输出列表 u。

程序段 5-1 的执行结果如图 5-1 所示。

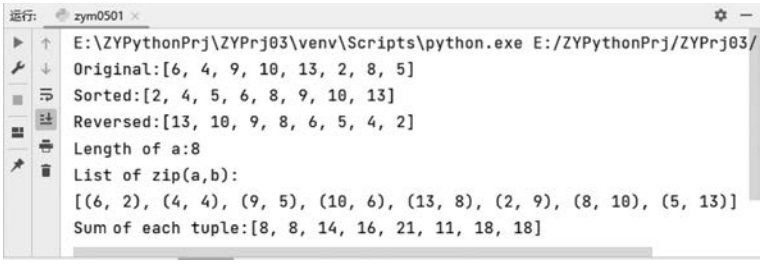

图 5-1　模块 zym0501 执行结果

5.2　自定义函数

内置函数主要实现通用数据处理功能或数学计算等，类似于其他计算机语言，Python 语言也提供了自定义函数。自定义函数在以下三方面具有内置函数不可替代的作用。

(1) 即使使用内置函数或引用包或模块中的函数可以实现所需要的功能，也需要自定义函数“包装”这些代码以增强程序的可读性。

(2) 在实际问题中总有一些专用的任务,需要手动编写自定义函数以解决这类问题。

(3) Python 语言程序文件(或模块)的标准结构是这样的:

```
自定义类 1
…… 
自定义类 m
自定义函数 1
…… 
自定义函数 n
if __name__ == '__main__':
    语句 1
    …… 
    语句 k
```

尽管可以直接在文件书写语句,但实际上 Python 语言程序具有上述的标准结构,即在"if __name__==='__main__':"上面只应定义"自定义类"和"自定义函数",而在"if __name__==='__main__':"内部(即其下面)才可写执行语句。可见,自定义函数几乎会出现在每一个 Python 程序中,用于将实现基本功能单位的代码组织在一起,以增强程序代码的可读性。

5.2.1　函数定义与调用

本书函数名的命名规则为全部使用小写英文字母和数字且以"my"开头,这是为了避免与 Python 语言的内置函数同名。在第 6 章介绍类时,将使用首字母大写的方式命名类名,也是为了避免与 Python 语言的内置类同名。

下面程序段 5-2 定义了三个简单的自定义函数,用来说明函数的定义与调用方法。

程序段 5-2　简单函数定义与调用实例


视频讲解

```
1    def myf():
2        print("Welcome.")
3    def myprint(s):
4        print(f'Output: {s}')
5    def mysum(a,b):
6        return a + b
7    if __name__ == '__main__':
8        myf()
9        myprint('Hello world.')
10       a = mysum(3,5)
11       myprint(a)
```

程序段 5-2 的执行结果如图 5-2 所示。

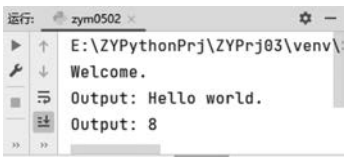

图 5-2　模块 zym0502 执行结果

在程序段 5-2 中定义了三个函数,其一为无参数无返回值的函数 myf;其二为带有一个参数但无返回值的函数 myprint;其三为有参数有返回值的函数 mysum。

定义函数头的语法为:"def　函数名(参数列表)",函数头下所有缩进的语句均属于该函数。参数列表可以为空,也可以为多个参数,定义函数头时的参数称为形式参数,简称形参;调用函数时传递给形参的参数值,称为实际参数,简称实参。函数的返回值借助 return 语句实现。函数的标准形式如下:

```
def 函数名(形式参数):
    语句 1
    ……
    语句 n
    return 表达式
```

回到程序段 5-2 中,第 1、2 行定义了一个无参数无返回值的函数 myf。第 1 行“def myf():”为函数头,用关键字“def”开头(def 是“define”的缩写,表示“定义”),不管函数有没有参数,函数名后的括号不能少,括号“()”是函数的特征,后面跟上“:”号。函数 myf 只有一条语句,即第 2 行“print("Welcome.")”,输出“Welcome.”。

在第 8 行“myf()”调用了函数 myf,调用函数时,即使函数没有参数,圆括号“()”也不能省略。这是无参数无返回值类型的函数的调用方法,即使用形如“函数名()”的形式调用。

在程序段 5-2 中,第 3、4 行定义了一个带有一个参数且无返回值的函数 myprint。第 3 行“def myprint(s):”定义函数头,对比第 1 行的函数 myf 的定义,这里在圆括号内部加入了参数 s。函数 myprint 只有一条语句,即第 4 行“print(f'Output:{s}')”输出参数 s 的值。

第 9 行“myprint('Hello world.')”调用自定义函数 myprint 输出“Output: Hello world.”。这是带参数无返回值类型的函数的调用方法,即使用形如“函数名(实参)”的形式调用。

回到第 5、6 行,这里定义了一个带有两个参数和具有返回值的函数 mysum。

第 5 行“def mysum(a,b):”定义函数头,函数名为“mysum”,具有两个形式参数 a 和 b。该函数只有一条语句,即第 6 行“return a+b”返回 a 与 b 的和。

第 10 行“a=mysum(3,5)”调用 mysum,并将返回值赋给 a,这里的 a 与第 5、6 行的 a 无关。第 5、6 行的 a 的作用范围(称为作用域)是第 5、6 行的函数 mysum,可以视其为 mysum 的局部变量。第 10 行的 a 的作用域为第 10 行(定义开始)及其后续语句。

第 11 行“myprint(a)”调用 myprint 自定义函数输出 a,得到“Output: 8”。

下面的程序段 5-3 实现了一元二次方程求根运算。


视频讲解

程序段 5-3　文件 zym0503

```
1    import math
2    def mysqrt(a):
3        if a<0:
4            return math.sqrt(-a)*1j
5        else:
6            return math.sqrt(a)
7    def myequ(a,b,c):
8        if abs(a)<10e-8:
9            print('Not a quadratic equation.')
10           x1=0
11           x2=0
12       else:
13           t=mysqrt(b**2-4*a*c)
14           x1=(-b+t)/(2*a)
15           x2=(-b-t)/(2*a)
16       return (x1,x2)
17   if __name__=='__main__':
18       (a,b,c)=(3,-4,2)
19       so=myequ(a,b,c)
```

```
20          print(f'Solution of {a}x^2{b:+}x{c:+}:\n(x1,x2) = ({so[0]:.4f},{so[1]:.4f})')
```

在程序段 5-3 中，第 1 行“import math”装载模块 math，为第 4、6 两行的 sqrt 提供函数定义。

第 2～6 行为自定义函数 mysqrt，具有一个参数 a，返回这个参数的平方根。

第 7～16 行为自定义函数 myequ，具有三个参数 a、b 和 c，表示一元二次方程 $ax^2+bx+c=0$ 的系数。第 8 行“if abs(a)＜10e－8：”当 a 非常小时，认为其为 0，则第 9 行“print('Not a quadratic equation.')”输出非一元二次方程提示信息，并将 x1 和 x2 均赋为 0。否则（第 12 行“else：”），第 13 行“t＝mysqrt(b ** 2－4 * a * c)”调用自定义函数 mysqrt 计算 $\sqrt{b^2-4ac}$，赋给 t。第 14 行“x1＝(－b＋t)/(2 * a)”计算第一个根 x1 的值；第 15 行“x2＝(－b－t)/(2 * a)”计算第二个根 x2 的值。第 16 行“return (x1,x2)”输出两个根。

第 18 行“(a,b,c)＝(3,－4,2)”为 a、b 和 c 设定值。第 19 行“so＝myequ(a,b,c)”调用自定义函数 myequ 计算方程的根，赋给 so。第 20 行“print(f'Solution of {a}x^2{b：＋}x{c：＋}：\n(x1,x2) ＝ ({so[0]：.4f},{so[1]：.4f})')”输出方程的两个根。

程序段 5-3 的执行结果如图 5-3 所示。

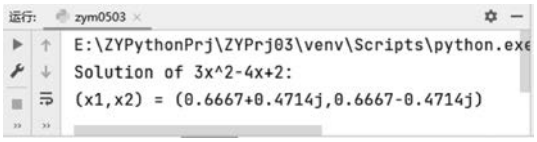

图 5-3　模块 zym0503 执行结果

由程序段 5-3 可得如下三条结论。

（1）自定义函数可以嵌套调用其他的自定义函数（或其本身）。当一个函数直接或间接调用其本身时，称为递归调用函数，见 5.3 节。

（2）每个“变量”（或“标签”）都有作用域和生存期两类特征。作用域是指该“标签”可以被访问的范围，例如，第 2 行的“a”的作用域为第 2～6 行的函数 mysqrt；而第 7 行的“a”的作用域为第 7～16 行的函数 myequ。这类“标签”可以称为局部标签，上述的两个“a”因其作用域不同，故互不影响。生存期是指从标签产生（在内存中建立“地位”）直到标签消失（从内存中“清除”）的生命期。函数中的局部标签的生存期一般为创建该标签至函数结束运行的这段时间，Python 语言内建了内存管理机制，可以不用考虑标签的生存期。

（3）函数的返回值可以借助元组等数据结构返回多个值。在 Python 中没有 C++语言的“指针”和“引用”，也无法借助参数返回值，但是可以借助元组、列表和字典等返回函数内部任意数量的计算结果。

此外，学过 C++语言的读者都知道 C++语言中有“函数重载”的定义，即函数名相同、函数参数个数不同或函数参数类型不同的多个函数可以并存在程序中，这种现象称为“函数重载”。在 Python 语言中没有这种函数重载的现象，因为 Python 没有“变量”的概念，函数的形式参数也是“标签”，每个形式参数“标签”在函数被用时，才由实际参数给它们赋值，那时才创建这些“标签”（并确定下各个“标签”的类型），所以无法借助“标签”实现函数的重载。

5.2.2　可变参数函数

函数可以有 0 个或多个参数，在调用函数时，实际参数的个数要与形式参数的个数相

同。但有两种情况,实际参数的个数可以与形式参数的个数不同,即带默认参数的情况和不定长参数的情况。

下面首先介绍默认参数的情况。程序段 5-4 是两个数的求和函数,带有三个参数,其中一个参数具有默认值。


视频讲解

程序段 5-4　参数带默认值的程序实例

```
1    def mysum(a,b = 0,c = True):
2        if c:
3            d = a + b
4        else:
5            d = abs(a) + abs(b)
6        return d
7    if __name__ == '__main__':
8        (a,b) = (3, - 5)
9        u1 = mysum(a,b)
10       print(f'u1 = {u1}')
11       u2 = mysum(a,b,False)
12       print(f'u2 = {u2}')
13       u3 = mysum(c = True,a = - 3)
14       print(f'u3 = {u3}')
15       u4 = mysum(c = False,b = 12, a = - 3)
16       print(f'u4 = {u4}')
```

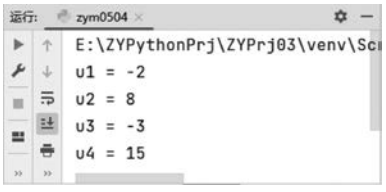

图 5-4　模块 zym0504 执行结果

程序段 5-4 的输出结果如图 5-4 所示。

在程序段 5-4 中,第 1～6 行定义了函数 mysum,该函数具有三个参数,其中,参数 a 为普通参数;参数 b 设定了默认值 0,用“b=0”表示;参数 c 设定了默认值 True,用“c=True”表示。

注意,设定了默认值的参数必须位于没有设定默认值的普通参数的右边。例如,在第 1 行中,如果给参数 a 设定了默认值,而参数 b 没有设默认值,那这种做法就是错误的。设定了默认值的参数,在调用函数时可以不为这些参数指定实参,此时将使用这些参数的默认值。

例如,在第 9 行“u1=mysum(a,b)”调用 mysum 时,省略了参数 c,此时参数 c 将使用默认值 True。第 13 行“u3=mysum(c=True,a=－3)”调用 mysum 时,省略了参数 b,此时 b 将使用默认值 0。

回到程序段 5-4 第 1～6 行的函数 mysum。

第 2～6 行为函数 mysum 的函数体。第 2 行“if c:”判断如果 c 为真,则执行第 3 行“d=a+b”计算 a 与 b 的和,赋给 d;否则(第 4 行),第 5 行“d=abs(a)+abs(b)”计算 a 的绝对值与 b 的绝对值的和,赋给 d。第 6 行“return d”返回 d。

第 8 行“(a,b)=(3,－5)”将 3 赋给 a,将－5 赋给 b。第 9 行“u1=mysum(a,b)”调用自定义函数 mysum 将实参 a 赋给形参 a,将实参 b 赋给形参 b。第 10 行“print(f'u1 = {u1}')”输出 u1。

第 11 行“u2=mysum(a,b,False)”调用自定义函数 mysum,按实参与形参的位置对应关系,将实参 a 赋给形参 a,将实参 b 赋给形参 b,将 False 赋给形参 c。第 12 行“print(f'u2 = {u2}')”输出 u2。

第 13 行“u3＝mysum(c＝True,a＝－3)”调用自定义函数 mysum,这里的参数表中使用了形如“形参名＝”的形式,这种形式称为关键字参数。关键字参数根据形参名将实参赋给形参,无须关心形参与实参的位置对应关系。这里,参数 b 使用默认值 0。第 14 行“print(f'u3 ＝ {u3}')”输出 u3。

第 15 行“u4 ＝ mysum(c＝False,b＝12, a＝－3)”借助关键字参数将实参赋给形参的方法调用自定义函数 mysum。第 16 行“print(f'u4 ＝ {u4}')”输出 u4。

由程序段 5-4,可以得出以下结论。

(1) 默认值参数只能在非默认值参数的右边,即默认值参数的右边不能出现非默认值参数。默认值参数可以有多个,甚至可以将全部参数设为默认值参数。在调用函数时,如果不指定默认值参数,将使用默认值参数定义时的默认值。

(2) 在调用函数时,不指定参数名(即不指定“参数名＝”这种形式)时,实参和形参是靠位置对应关系进行传递的,即处于第 n 个位置的实参将传递给第 n 个位置的形参。例如,mysum(3,－5)将 3 传递给 a,将－5 传递给 b。

(3) 在给定关键字参数时,关键字参数前的参数仍然靠位置对应关系传递,关键字参数之后的参数必须均为关键字参数。例如,“mysum(－3,c＝True)”将－3 传递给 a,关键字参数“c＝True”将 True 传递给 c,参数 b 使用默认参数 0;调用形式“mysum(－3,b＝12,False)”是错误的,由于“b＝12”是关键字参数,后面的“True”是普通参数,故是错误的,关键字参数后面的参数必须全用关键字参数;“mysum(－3,8,c＝False)”这里关键字参数“c＝False”前面的两个实参按位置对应关系传递给形参,故－3 传递给 a,8 传递给 b。

(4) 在给定关键字参数时,关键字参数不受参数位置的限制,但是关键字参数出现后的后续参数必须均为关键字参数。例如,“mysum(c＝False,b＝12,a＝－3)”是正确的,依靠关键字参数将 False 传递给形参 c、将 12 传递给形参 b、将－3 传递给形参 a;“mysum(－3,c＝True,b＝12)”这里的关键字参数“c＝True”前的参数仍然靠位置对应关系进行参数传递,即－3 传递给形参 a,而根据关键字参数(而不是位置对应关系)将 True 传递给形参 c、将 12 传递给形参 b。

注意:上述实例中,关键字参数不是必须使用的,但是在特殊情况(即下文的不定长参数情况)下,关键字参数是必须使用的。

现在,介绍第二种实参与形参个数不同的情况,即不定长参数情况。此时,实参的个数可以为任意多个,例如,内置函数 print 可以具有可变长度的参数。

与函数通过元组等可以返回多个值相似,可以通过元组或字典向函数传递多个值。在不定长参数的情况下,使用“＊形式参数名”作为参数可将输入的多个参数“打包”为一个元组,使用“＊＊形式参数名”作为参数可将输入的多个参数“打包”为一个字典。这样,使用不定义长参数的函数头的语法为:

```
def 函数名(普通参数表1, *可变长参数, 普通参数表2)
def 函数名(普通参数表1, **可变长参数)
```

在上述语法中,“普通参数表 1”根据需要可有可无。但是如果设定了“普通参数表 2”,在调用函数时必须使用关键字参数,否则给定的参数将被“＊可变长参数”“吞吃”,即被认为是可变长参数的一部分。在“def　函数名(普通参数表 1, ＊＊可变长参数)”中“可变长参数”后不能再有任何参数。

程序段 5-5 展示了可变长参数的设计方法。


视频讲解

程序段 5-5　文件 zym0505

```
1    def mysum1(*n):
2        print('n=',n)
3        print('The numbers:')
4        for e in n:
5            print(e,end=' ')
6        print()
7        print('have sum:',end='')
8        print(sum(n))
9    def mysum2(a,*n,b):
10       if b:
11           c=sum(n)
12       else:
13           c=a*sum(n)
14       return c
15   def mysum3(h,**n):
16       print('n=',n)
17       s=0
18       for e in n.values():
19           if h:
20               s+=e
21           else:
22               s+=2*e
23       return s
24   if __name__=='__main__':
25       mysum1(1,2,3,4,5,6,7,8,9,10)
26       c1=mysum2(2,1,2,3,4,5,6,7,8,9,10,b=False)
27       print(f'c1={c1}')
28       c2=mysum2(2,*(1,2,3,4,5,6,7,8,9,10),b=False)
29       print(f'c2={c2}')
30       c3=mysum3(False,a=1,b=2,c=3,d=4)
31       print(f'c3={c3}')
32       c4=mysum3(False,**{'a':1,'b':2,'c':3,'d':4})
33       print(f'c4={c4}')
```

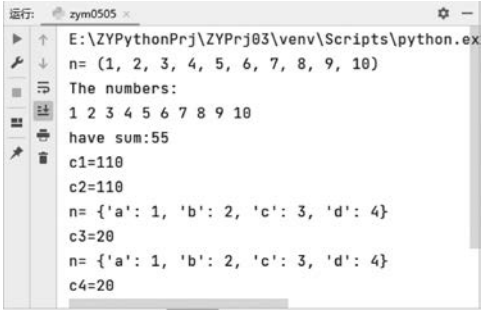

图 5-5　模块 zym0505 执行结果

程序段 5-5 的执行结果如图 5-5 所示。

在程序段 5-5 中，第 1～8 行定义了函数 mysum1，具有一个可变长参数“*n”。第 2 行“print('n=',n)”输出 n，结合第 25 行和图 5-5 可知，参数 n 是输入的实参“打包”成的元组，即“n=（1，2，3，4，5，6，7，8，9，10）”。第 3 行“print('The numbers:')”输出提示信息。第 4、5 行为一个 for 结构，输出（元组）n 中的所有元素，即输出全部的实参。第 6 行“print()”输出空行。第 7 行“print('have sum:',end='')”输出提示信息“have sum:”，第 8 行“print(sum(n))”输出（元组）n 中全部元素的和。

第 9～14 行定义了函数 mysum2，包括一个普通参数 a、可变长参数 n 和一个普通参数 b。第 10～13 行为一个 if-else 结构，如果 b 为真，则第 11 行“c=sum(n)”计算（元组）n 的全

部元素的和，赋给 c；否则，第 13 行“c＝a * sum(n)”计算(元组)n 的全部元素的和，并乘以 a 后赋给 c。第 14 行“return c”返回 c。注意，在调用函数 mysum2 时，参数 b 必须使用关键字参数调用，即用形如“b＝True”的形式调用。

第 15～23 行定义了函数 mysum3，包括一个普通参数 h 和可变长参数 n，这里的“ ** n”将输入的实参“打包”成字典，“ ** n”后面不能再有任何形式参数。第 16 行“print('n＝',n)”输出字典形式的 n，结合图 5-5 和第 30 行，将输出“n＝ {'a'：1， 'b'：2， 'c'：3， 'd'：4}”。第 17 行“s＝0”将 0 赋给 s。第 18～22 行为一个 for 结构，如果 h 为真，则第 20 行“s＋＝e”将字典 n 中各个元素的值累加到 s 中；如果 h 为假，则第 22 行“s＋＝2 * e”将字典 n 中各个元素的值的 2 倍累加到 s 中。第 23 行“return s”返回 s。

第 25 行“mysum1(1,2,3,4,5,6,7,8,9,10)”调用函数 mysum1，这里将 10 个参数传递给函数 mysum1 的形参“ * n”。第 26 行“c1＝mysum2(2,1,2,3,4,5,6,7,8,9,10,b＝False)”将 2 传递给形参 a；将“1,2,3,4,5,6,7,8,9,10”传递给形参“ * n”，此后，n 为元组(1,2,3,4,5,6,7,8,9,10)；借助关键字参数将“False”传递给形参 b。也可以将这些参数“1,2,3,4,5,6,7,8,9,10”用元组来表示，如第 28 行“c2＝mysum2(2， *(1,2,3,4,5,6,7,8,9,10)， b＝False)”所示，这里，是将“ *(1,2,3,4,5,6,7,8,9,10)”传递给“ * n”，第 28 行等价于第 26 行。注意，这里的“b＝False”只能用关键字参数，不能直接写成“False”。

第 30 行“c3＝mysum3(False,a＝1,b＝2,c＝3,d＝4)”调用函数 mysum3，将 False 传递给形参 h，将“a＝1,b＝2,c＝3,d＝4”传递给形参“ ** n”，此后，n 为字典“n＝ {'a'：1， 'b'：2， 'c'：3， 'd'：4}”。也可以直接用字典作为参数，如第 32 行“c4＝mysum3(False， ** {'a'：1,'b'：2,'c'：3,'d'：4})”所示，这里，将“ ** {'a'：1,'b'：2,'c'：3,'d'：4}”传递给“ ** n”，即 n 为字典“{'a'：1， 'b'：2， 'c'：3， 'd'：4}”。

最后，除了借助形如“ * n”和“ ** n”的形参实现函数可变参数输入外，还可以通过列表的形式实现函数可变输入数据(这种情况下，参数个数不变)，如程序段 5-6 所示。

程序段 5-6　列表作为函数参数


视频讲解

```
1    def mysum(v):
2        n = len(v)
3        s = 0
4        for e in v:
5            s += e ** 2
6        return s
7    if __name__ == '__main__':
8        v1 = [1,2,3,4,5,6,7,8,9]
9        s1 = mysum(v1)
10       print(f'v1 = {v1}')
11       print(f's1 = {s1}')
12       v1.append(10)
13       s2 = mysum(v1)
14       print(f'v1 = {v1}')
15       print(f's2 = {s2}')
```

在程序段 5-6 中，第 1～6 行定义了函数 mysum，有一个参数 v(视为列表)，该函数计算列表中全部元素的平方和。在第 8 行中“v1＝[1,2,3,4,5,6,7,8,9]”，第 9 行“s1＝mysum(v1)”调用 mysum 计算 v1 中全部元素的平方和；然后，第 12 行“v1.append(10)”在 v1 中添加元素 10，第 13 行“s2＝mysum(v1)”计算 v1 中全部元素的平方和。对比第 9 行，这里实际是

多处理了一个元素 10。表面上实参和形参的形式统一，但实际上，输入的数据个数是可变的。因此借助列表等数据结构作为函数参数，实际上可以实现输入的数据个数可变的函数。

程序段 5-6 的执行结果如图 5-6 所示。

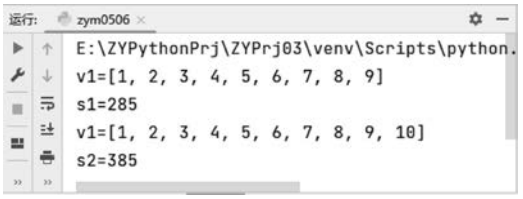

图 5-6　模块 zym0506 执行结果

可以用类似于程序段 5-6 的方法，借助元组和字典实现函数的可变输入数据(这种情况下，函数的参数个数不变，只是参数中的数据个数可变)。

5.2.3　函数返回值与变量作用域

函数的三个要素为函数名、输入参数和返回值。函数内部的处理过程就是普通的语句和算法。在前述内容中，已经体现了函数返回值的实现方法。这里再次强调，借助列表、元组和字典等可以实现函数返回多个数值。结合 5.2.2 节可知，借助元组、字典和列表，可以实现函数的可变参数(或可变数据)输入和多数据返回。

本节将表示数据的“标签”称为“变量”，变量有两个需要关心的特征，即作用域和生存期，前述 5.2.1 节也已经讨论过。这里再进一步讨论变量的作用域，主要有以下规律。

(1) 形参变量的作用域为函数本身，也就是说形参只在函数内部是可见的，可以使用，在其他地方不可使用。例如：

```
def mysqr(a):
    return a ** 2
```

这里的“a”的作用域为 mysqr 函数本身。

(2) Python 程序按照缩进关系确定程序段间的归属关系，在上一层次定义的变量，可以在本层或其下层次中的语句中使用，即上一层的变量在本层和其低级的层次中是可见的。但是，其低级中定义的同名变量，将“覆盖”上一层级的同名变量的可见性，即低级中有与上一级同名的变量，在这一低范围的语句中优先使用。

例如：

```
a = 3
b = 5
def mysum(x,y):
    a = 10
    z = a + b + x + y
    return z
print(mysum(11,12))
```

这里，在 mysum 内部的 a 为局部变量，而外部的“a＝3”可视为上一级变量(即全局变量)，这里的“a＝10”将“覆盖”全局的“a＝3”，而 b 将直接使用全局的 b，所以，输出结果为 38。注意：这里的“a＝10”不会改变全局的“a＝3”，如果想让“a＝10”改变全局的 a，需要在

mysum 内部“a=10”前面将 a 声明为全局的 a，即添加语句“global a”（这不是一个明智的选择，实际上，所有的程序中都应该尽可能避免使用全局变量，因为它们的作用域“太大”了，全局变量的修改会造成“牵一发而动全身”的不良后果）。

（3）在低级中出现的变量，尽可能不在上一级直接使用（尽管若上一级没有同名变量时，使用这些低级变量并不会导致执行错误）。如果低级中出现的变量需要在其上一级中使用，就在上一级中定义同名变量。例如：

```
a = 3
if a > 1:
    b = 5
print(a + b)
```

这里的 b 出现在 if 结构中，而 print(a+b)为“b=5”的上一级，其中使用了 b。这种做法不妥，应在“if a>1:”前添加一条“b=0”之类的初始化语句（注意，添加了“b=0”后，if 结构中的 b 将具有和“b=0”中的 b 不同的内存地址）。

下面通过程序段 5-7 进一步分析同名变量的作用域。

程序段 5-7　文件 zym0507


```
1   def mysum(v):
2       def mysqr(a):
3           return a ** 2
4       def mycub(a):
5           return a ** 3
6       s1 = s2 = s3 = 0
7       for a in v:
8           s1 += a
9           s2 += mysqr(a)
10          s3 += mycub(a)
11      return (s1,s2,s3)
12  if __name__ == '__main__':
13      a = [1,2,3,4,5]
14      v = mysum(a)
15      print(f'v = {v}')
```

在程序段 5-7 中，第 1～11 行定义了函数 mysum，其中第 2、3 行定义了函数 mysqr，计算其参数的平方值；第 4、5 行定义了函数 mycub，计算其参数的立方值。

第 6 行“s1=s2=s3=0”将 s1、s2 和 s3 均初始化为 0。第 7～10 行为一个 for 结构，其中调用了函数 mysqr 和 mycub，循环结构将 v 的全部元素的和赋给 s1，将 v 的各个元素的平方和赋给 s2，将 v 的各个元素的立方和赋给 s3。第 11 行“return (s1,s2,s3)”以元组的形式返回计算结果（这里的圆括号可以省略）。

第 13 行“a=[1,2,3,4,5]”定义列表 a；第 14 行“v=mysum(a)”调用 mysum 将计算结果赋给 v；第 15 行“print(f'v={v}')”输出 v，得到“v=(15, 55, 225)”。

5.2.4　函数闭包与装饰器

由前述可知，函数内部可以嵌套新的函数定义。有一种特殊函数嵌套定义方式，如下所示：

```
def 函数名 1(参数表 1)
```

```
语句 1
语句 2
……
语句 m
def 函数名 2(参数表 2)
    语句 t1
    语句 t2
    ……
    语句 tk
语句 m+1
语句 m+2
……
语句 n
return 函数名 2
```

上述函数定义是“奇怪”的,因为外层函数(函数名 1)的返回值是内层函数(函数名 2),同时要求内层函数中使用外层函数的参数(参数表 1 中的部分或全部参数)或变量(当参数表 1 为空时,内层函数必须使用外层函数中的变量)。这种形式的函数称为函数闭包(准确地讲,内层函数是闭包)。上述的“语句 1”至“语句 m”和“语句 m+1”至“语句 n”是可选的,但是需保证内层函数中使用了外层函数的参数或变量。如果“参数表 1”为空,那么“语句 1”至“语句 m”不能为空。

下面通过程序段 5-8 介绍函数闭包。


视频讲解

程序段 5-8　函数闭包实例

```
1   def mysum(v,h):
2       t=1
3       if h:
4           t=2
5       def mys(u):
6           s=0
7           for e in v:
8            s+=u*t*e
9           return s
10      return mys
11  if __name__=='__main__':
12      v=[1,2,3,4,5]
13      u=3
14      s=mysum(v,True)(u)
15      print(f's={s}')
```

在程序段 5-8 中,第 1～10 行定义了函数 mysum,具有 v 和 h 两个参数;第 2 行“t=1”令 t 为 1;第 3、4 行为一个 if 结构,如果 h 为真,则赋 t 为 2。

第 5～9 行为闭包函数 mys,具有一个参数 u。其中,第 6 行“s=0”设 s 为 0;第 7、8 行为一个 for 结构,将 v 的每个元素倍乘 u*t 后累加到 s 中。第 9 行“return s”。

第 10 行“return mys”表示外层函数 mysum 返回内层的函数 mys,从而构成了函数闭包。

第 12 行“v=[1,2,3,4,5]”定义 v;第 13 行“u=3”定义 u。

第 14 行“s=mysum(v,True)(u)”为函数闭包的调用,注意这里有两对圆括号,第一对圆括号中的“v,True”作为外层函数的参数;第二对圆括号中的“u”作为内层函数的参数。

第 15 行“print(f's={s}')”输出结果“s=90”。

这种类似于程序段 5-7 中的函数闭包只有两个函数，可称为两级函数闭包。可以实现多级函数闭包，前提是实现的功能确实需要那么做。对于三级函数闭包，在调用函数时需要使用三对圆括号（没有参数时，圆括号也不能少）。

现在在程序段 5-8 的基础上实现一个三级函数闭包，如程序段 5-9 所示。

程序段 5-9　三级函数闭包实例


```
1    def mysum(v,h):
2        t = 1
3        if h:
4            t = 2
5        def mys(u):
6            s = 0
7            for e in v:
8             s += u * t * e
9            def mys2(p):
10               s2 = p * s
11               return s2
12           return mys2
13       return mys
14   if __name__ == '__main__':
15       v = [1,2,3,4,5]
16       u = 3
17       s = mysum(v,True)(u)(2)
18       print(f's = {s}')
```

程序段 5-9 在程序段 5-8 的基础上，添加了第 9～11 行的语句，并修改了第 12 行语句。注意第 17 行语句“s=mysum(v,True)(u)(2)”是对这个三级函数闭包的调用方法，第 18 行语句的执行将得到结果“s=180”。

有一种特殊的闭包，类似于数学上的复合函数调用（5.4 节将介绍 Python 语言内置的复合函数调用方法），称为函数装饰器，简称装饰器。在程序设计过程中会遇到这种情况，即有一个设计良好的函数（不妨称其为被“装饰”的函数），想为其添加一些功能，新添加的这些功能不能改变函数的功能，新添加的功能可用于输出一些提示性信息，也可对参数进行数学运算，只是这些计算结果与被“装饰”的函数无关。

例如，有一个设计良好的函数 mysum(v,h)，想为其添加一些提示性信息，可以定义一个闭包函数，函数名为 mysumex(f)，只能有一个形式参数，这里定义为 f，调用时，将 mysum 函数赋给 f。在此，作为装饰器的闭包函数的形式为：

```
def mysumex(f):
    def 函数名(参数表)                    #注意,"函数名"可随意取合法的名称,但是"参数表"
                                          #必须与 mysum 函数相同
        语句 1
        ……
        语句 m                            #语句 1 至语句 m 是可选的
        f(参数表)
        语句 m+1
        ……
        语句 n                            #语句 m+1 至语句 n 是可选的
return 函数名
```

```
@mysumex                          #使用"@"符号加上闭包函数名
def mysum(参数表)
    语句 1
    ……
    语句 k
```

注意：上述代码中：①闭包函数的外层函数名“mysumex”必须和“@mysumex”的“装饰器”声明处的名称相同；②闭包函数的内层“函数名”和“return 函数名”处的“函数名”必须相同，以构成闭包，这个“函数名”可以为任意合法的标识符；③装饰器 mysumex 只能有一个参数，这个参数将传递被“装饰”的函数 mysum；④闭包函数的内层函数的“参数表”必须与被“装饰”的函数相同；⑤调用装饰器时，仍然调用被“装饰”的函数 mysum，但是会执行作为装饰器的闭包函数 mysumex，相当于给被“装饰”的 mysum 函数添加了新的功能；⑥闭包函数 mysumex 中的“语句 1”至“语句 m”和“语句 m＋1”至“语句 n”在遇到“@mysumex”时被执行，而不是调用被“装饰”的函数时才执行，这一点尤其要注意。

下面程序段 5-10 演示了装饰器的用法。


视频讲解

程序段 5-10　装饰器用法实例

```
1    def mysumex(f):
2        print('This is begin of mysumex.')
3        def mydef(v,h):
4            print(f'v = {v}')
5            print(f'h = {h}')
6            s = f(v,h)
7            print(f's = {s}')
8            return s
9        print('This is end of mysumex.')
10       return mydef
11   @mysumex
12   def mysum(v,h):
13       s = sum(v)
14       if h:
15           s = 2 * s
16       return s
17   if __name__ == '__main__':
18       s = mysum([1,2,3,4,5],True)
19       print(f'In main program, s = {s}')
```

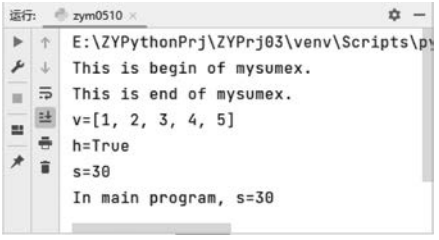

图 5-7　模块 zym0510 执行结果

程序段 5-10 的执行结果如图 5-7 所示。

在程序段 5-10 中，第 1～10 行定义了一个闭包函数，函数名为 mysumex，具有一个参数 f。第 11 行将第 12 行的 mysum 定义为装饰器。第 12～16 行定义函数 mysum，具有两个参数 v 和 h。

在第 12～16 行的函数 mysum 中，第 13 行“s＝sum(v)”计算 v 中全部元素的和，赋给 s；第 14～15 行为一个 if 结构，如果 h 为真，则第 15 行“s＝2 * s”将 s 的值翻倍。第 16 行“return s”返回 s。

注意：程序段 5-10 在执行时，第 1～10 行为函数定义，不执行；第 12～16 行为函数定义，不执行，但是第 11 行“@mysumex”定义装饰器，将执行装饰器函数 mysumex，此时输出

提示信息："This is begin of mysumex."和"This is end of mysumex."，如图 5-7 所示。然后，执行第 17～19 行。

第 18 行"s＝mysum([1,2,3,4,5],True)"调用 mysum 函数时，将调用装饰器函数 mysumex 中的 mydef，而不再执行函数 mydef 外部的任何语句。接着，第 4 行"print(f'v＝{v}')"输出"v＝[1, 2, 3, 4, 5]"；第 5 行"print(f'h＝{h}')"输出"h＝True"；第 6 行"s＝f(v,h)"用被装饰的函数 mysum 替换 f，即执行真实的函数 mysum(第 12～16 行)的代码，得到 s；第 7 行"print(f's＝{s}')"输出"s＝30"，从被调函数中返回。最后，第 19 行"print(f'In main program, s＝{s}')"输出"In main program, s＝30"。

程序段 5-10 是典型的装饰器的用法。一个装饰器可以"装饰"多个函数，要求所有被"装饰"的函数必须具有与装饰器函数闭包中内层函数相同的参数表。

下面程序段 5-11 是在程序段 5-10 的基础上，使用装饰器 mysumex"装饰"三个函数的例子。

程序段 5-11　装饰器"装饰"多个函数的实例


视频讲解

```
import math
def mysumex(f):
    print('This is begin of mysumex.')
    def mydef(v,h):
        print(f'v = {v}')
        print(f'h = {h}')
        s = f(v,h)
        print(f's = {s}')
        return s
    print('This is end of mysumex.')
    return mydef
@mysumex
def mysum(v,h):
    s = sum(v)
    if h:
        s = 2 * s
    return s
@mysumex
def mymul(v,h):
    s = 1
    for e in v:
        s * = e
    if h:
        s = s/2
    return s
@mysumex
def mydis(v,h):
    s = 0
    for e in v:
        s += e ** 2
    if h:
        s = math.sqrt(s)
    return s
if __name__ == '__main__':
    s1 = mysum([1,2,3,4,5],True)
    print(f'In main program, s1 = {s1}')
```

```
37          s2 = mymul([1, 2, 3, 4, 5], True)
38          print(f'In main program, s2 = {s2}')
39          s3 = mydis([1, 2, 3, 4, 5], True)
40          print(f'In main program, s3 = {s3:.4f}')
```

程序段 5-11 是在程序段 5-10 的基础上添加了第 18～25 行、第 26～33 行、第 37～40 行。

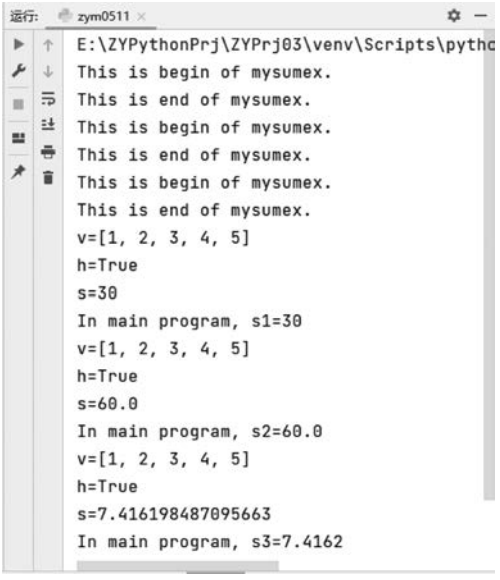

图 5-8　模块 zym0511 执行结果

第 18～25 行将装饰器函数 mysumex“装饰”函数 mymul，这里的函数 mymul 计算参数 v 中各个元素的积(第 20～22 行)，赋给 s，如果参数 h 为真，则将 s 的值减半。第 25 行“return s”返回 s。

第 26～33 行将装饰器函数 mysumex“装饰”函数 mydis，这里的函数 mydis 计算参数 v 中各个元素的平方和(第 28～30 行)，赋给 s，如果参数 h 为真，则将 s 的值求算术平方根。第 33 行“return s”返回 s。

在执行程序段 5-11 时，将首先执行第 12 行、第 18 行和第 26 行的装饰器定义，输出 3 次提示信息“This is begin of mysumex.”和“This is end of mysumex.”，如图 5-8 所示。然后，执行第 34～40 行的可执行语句。

程序段 5-11 的执行结果如图 5-8 所示。

由程序段 5-11 和图 5-8 可知，如果不想装饰器在定义时输出信息，则在设计装饰器闭包函数时，外层函数和内层函数间不放任何语句即可。

5.3　递归函数

Python 语言支持递归函数。所谓递归函数是指可以直接或间接调用函数本身的函数。下面以求阶乘为例介绍递归函数的设计方法。

阶乘的定义为：$n! = n \times (n-1) \times \cdots \times 2 \times 1 = n \times (n-1)!$。若定义阶乘函数为 myfac，则求 n 的阶乘的计算方式为 myfac(n)=n * myfac(n−1)，即 myfac 将调用 myfac 函数，这称为递归调用。递归调用必须有终止条件，对于阶乘而言，其终止条件为 0!=1 或 1!=1，即 myfac(0)=myfac(1)=1。

设计递归函数的方法为：

(1) 首先使用 if 结构编写递归函数的终止条件；

(2) 编写递归调用算法。

下面程序段 5-12 介绍了阶乘的递归函数设计方法。


视频讲解

程序段 5-12　实现阶乘的递归函数

```
1    def myfac(n):
2        if n == 0 or n == 1:
```

```
3            return 1
4        return n * myfac(n - 1)
5    if __name__ == '__main__':
6        n = int(input('Please input an integer (n >= 0):'))
7        m = myfac(n)
8        print(f'n!= {m}')
```

在程序段 5-12 中，第 1～4 行定义了阶乘函数 myfac。第 1 行“def myfac(n):”为函数头，具有一个参数 n。第 2、3 行为递归函数 myfac 的终止条件，即当 n 等于 0 或 1 时，返回 1。第 4 行“return n * myfac(n－1)”为递归算法，返回递归调用后的值。

第 6 行“n＝int(input('Please input an integer (n＞＝0)：'))”输入非负整数 n。第 7 行“m＝myfac(n)”调用 myfac 函数，计算 n 的阶乘，结果赋给 m。第 8 行“print(f'n!＝{m}')”输出 n! 的值。

程序段 5-12 的执行结果如图 5-9 所示。

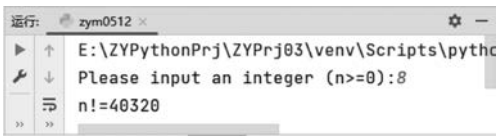

图 5-9　模块 zym0512 执行结果

由程序段 5-12 可知，递归函数的设计要旨如下。

(1) 设计终止条件。例如，程序段 5-12 中的求阶乘函数 myfac 的第 2、3 行，表示其终止条件为 n 等于 0 或 1 时，返回 1。

(2) 设计递归调用。例如，在程序段 5-12 中，求阶乘的递归调用为“return n * myfac(n－1)”，即返回递归调用后的值。

递归调用过程中将出现大量的局部变量，需要占用较多的栈空间。

下面的程序段 5-13 使用递归函数求 Fibonacci 数。Fibonacci 数的规律为：

$$\begin{cases} F_1 = F_2 = 1 \\ F_n = F_{n-1} + F_{n-2}, n > 2 \end{cases}$$

程序段 5-13　利用递归函数求 Fibonacci 数的第 n 项


```
1    def myfib(n):
2        if n == 1 or n == 2:
3            return 1
4        return myfib(n - 1) + myfib(n - 2)
5    if __name__ == '__main__':
6        n = int(input('Please input an integer (n > 0):'))
7        m = myfib(n)
8        print(f'Fibonacci({n}) = {m}.')
```

在程序段 5-13 中，第 1～4 行为递归函数 myfib，具有一个参数 n。第 2、3 行为该递归函数的终止条件，即当 n 等于 1 或 2 时，返回 1。第 4 行“return myfib(n－1)＋myfib(n－2)”为递归算法，返回 myfib(n－1)与 myfib(n－2)的和。

第 6 行“n ＝ int(input('Please input an integer (n＞0)：'))”输入正整数 n。第 7 行“m＝myfib(n)”调用 myfib 函数，将计算结果赋给 m。第 8 行“print(f'Fibonacci({n})＝{m}.')”输出第 n 个 Fibonacci 数。

程序段 5-12 的执行结果如图 5-10 所示。

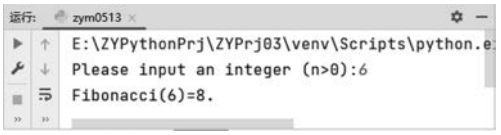

图 5-10　模块 zym0513 执行结果

递归函数还可以求解排列组合问题。例如,从 n 个人中随机选取 k 个人的组合数为

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$

用递归函数实现时,其终止条件为:当 $k=0$ 或 $k=n$ 时,返回 1。下面的程序段 5-14 实现了上述求组合数问题。


视频讲解

程序段 5-14　求组合数的递归函数实例

```
1    def mycom(n,k):
2        if k == 0 or k == n:
3            return 1
4        return mycom(n - 1,k - 1) + mycom(n - 1,k)
5    if __name__ == '__main__':
6        (n,k) = map(int,input('Please input two integers n,k(0 <= k <= n):').split(','))
7        m = mycom(n,k)
8        print(f'C(n,k) = {m}.')
```

在程序段 5-14 中,第 1～4 行为递归函数 mycom,具有两个参数 n 和 k。第 2、3 行为其终止条件,即当 k 等于 0 或 n 时,返回 1。第 4 行“return mycom(n－1,k－1)＋mycom(n－1,k)”为递归算法,返回 mycom(n－1,k－1)与 mycom(n－1,k)的和。

第 6 行“(n,k)＝map(int,input('Please input two integers n,k(0＜＝k＜＝n)：').split(','))”输入两个非负整数 n 与 k,要求 n 大于或等于 k,这里使用了 map 函数,将 int 函数映射到两个输入的字符串上,将在 5.4 节进一步介绍 map 函数。

第 7 行“m＝mycom(n,k)”调用递归函数 mycom,计算结果赋给 m。第 8 行“print(f'C(n,k)＝{m}.')”输出组合数(n,k)的值。

程序段 5-14 的执行结果如图 5-11 所示。

递归函数的经典实例为汉诺塔问题。如图 5-12 所示,有 A、B、C 三根柱子,A 柱上放置了 n 个盘子,这 n 个盘子大小互不相同,且从大至小向上依次堆叠。欲将 A 柱上的这 n 个盘子从 A 柱借助 B 柱移动到 C 柱上,要求每次只能移动一个盘子,且移动过程中,A、B、C 三根柱子上的盘子必须始终大盘在下、小盘在上。

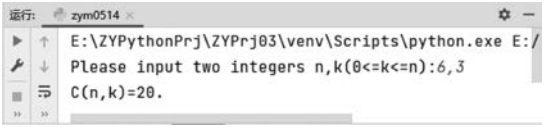

图 5-11　模块 zym0514 执行结果

图 5-12　汉诺塔问题

汉诺塔问题的终止条件为:当 A 柱上只有一个盘子时,将该盘从 A 柱移动到 C 柱。

汉诺塔问题的递归调用为:将 A 柱上的 $n-1$ 个盘子借助 C 柱移动到 B 柱;将 A 柱上的第 n 个盘子(最大的盘子)从 A 柱直接移动到 C 柱;将 B 柱上的 $n-1$ 个盘子借助 A 柱移动到 C 柱上。这样就将具有 n 个盘子的汉诺塔问题,转化为两个具有 $n-1$ 个盘子的汉诺

塔问题，从而实现了递归调用。

下面的程序段 5-15 解决了汉诺塔问题。

程序段 5-15　递归函数实现汉诺塔问题


```
1    def myhan(n,a,b,c):
2        if n == 1:
3            print(f'{a}--->{c}')
4            return
5        myhan(n-1,a,c,b)
6        print(f'{a}--->{c}')
7        myhan(n-1,b,a,c)
8    if __name__ == '__main__':
9        n = int(input('Please input the number of disks:'))
10       myhan(n,'A','B','C')
```

程序段 5-15 的执行结果如图 5-13 所示。

图 5-13　模块 zym0515 执行结果

在程序段 5-15 中，第 1～7 行定义了递归函数 myhan，具有四个参数 n、a、b 和 c。第 2～4 行为其终止条件，即当 A 柱上只剩一个盘子(n 等于 1)时，将该盘从 A 柱移动到 C 柱，返回。第 5～7 行为递归算法，第 5 行“myhan(n－1,a,c,b)”将 A 柱上的 n－1 个盘子从 A 柱借助 C 柱移动到 B 柱；第 6 行“print(f'{a}--->{c}')”将 A 柱上剩余的一个盘子直接移动到 C 柱；第 7 行“myhan(n－1,b,a,c)”将 B 柱上的 n－1 盘子借助 A 柱移动到 C 柱。

第 9 行“n = int(input('Please input the number of disks：'))”输入盘子数 n。第 10 行“myhan(n,'A','B','C')”调用 myhan 函数解决 n 个盘子的汉诺塔问题。

3.4 节介绍了两种排序方法，即冒泡法和选择法。这里介绍借助递归算法实现的快速排序算法，这种方法的执行效率非常高，与内置的 sort 方法效率相当。

快速排序法的实现原理如下：

(1) 随机从序列中选择一个数作为基准数，一般地，选择序列的首元素作为基准数。

(2) 在序列中从右向左找一个小于基准数的数，不妨将其索引号设为 j；然后，从左向右找一个大于基准数的数，不妨将其索引号设为 i。如果 i<j，则将这两个数交换位置。这样，实现了第一轮查找。

(3) 接着，从当前的 j 索引位置继续从右向左找一个小于基准数的数，找到后，仍将其索引号记为 j；然后，从当前的 i 索引位置从左向右找一个大于基准数的数，仍将其索引号记为 i。如果 i<j，则将这两个数交换位置；如果从左向右找的过程中，出现了 j=i，或者从左向右找(可能没有找到满足条件的元素)时出现了 i=j，则将基准数与第 j 个数互换，算法停止；或者，从左向右查找，如果发现了 j 对应着基准数，则算法停止。注意：每一轮查找中，总是先执行从右向左查找。

(4) 上述过程完成后,以基准数为分界点,序列将分成两个子序列,每个子序列再重复上述算法。直到所有的子序列均排序完成,则整个序列的排序完成。

下面程序段 5-16 使用递归方法实现了快速排序法。


视频讲解

程序段 5-16　快速排序法实例

```
1    def myquicksort(a,h,t):
2        if h>=t:
3            return
4        b=a[h]
5        i=h
6        j=t
7        while i!=j:
8            while a[j]>=b and j>i:
9                j-=1
10           while a[i]<=b and i<j:
11               i+=1
12           if j>i:
13               a[i],a[j]=a[j],a[i]
14       a[h]=a[i]
15       a[i]=b
16       myquicksort(a,h,i-1)
17       myquicksort(a,i+1,t)
18   if __name__=='__main__':
19       a=[6,10,16,3,8,23,15,7,2,11]
20       print(f'Original: {a}.')
21       myquicksort(a,0,len(a)-1)
22       print(f'Sorted: {a}.')
```

程序段 5-16 的执行结果如图 5-14 所示。

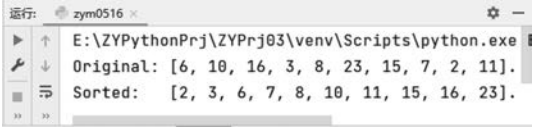

图 5-14　模块 zym0516 执行结果

在程序段 5-16 中,第 1～17 行为自定义函数 myquicksort,具有三个参数,a 表示排序的列表,h 表示列表的首索引号(为 0),t 表示列表的尾索引号(为 len(a)－1)。第 2、3 行为终止条件,即 h 与 t 相同时,终止。第 4～15 行为第一轮搜索,第 4 行“b＝a[h]”将首元素作为基准数;第 5 行“i＝h”将 h 赋给 i,即 i 指向最左边的元素;第 6 行“j＝t”将 t 赋给 j,即 j 指向最右边的元素。

第 7～13 行为两边的搜索过程,先从右向左搜,再从左向右搜。第 7 行“while i!＝j:”当 i 不等于 j 时,开始这一轮搜索:第 8、9 行为从右向左搜,找第一个比基准数小的元素,第 8 行“while a[j]＞＝b and j＞i:”判断 a[j]大于或等于 b 且 j 大于 i 时,第 9 行“j－＝1”表示继续向左找,第 8、9 行循环查找,直到找到一个比基准数小的元素(其索引号为 j)。第 10、11 行为从左向右搜,找第一个比基准数大的元素,第 10 行“while a[i]＜＝b and i＜j:”判断 a[i]小于或等于 b 且 i 小于 j 时,第 11 行“i＋＝1”表示继续向右找,第 10、11 行循环查找,直到找到一个比基准数大的元素(其索引号为 i)。第 12 行“if j＞i:”判断如果 j 大于 i,则第 13 行“a[i],a[j]＝a[j],a[i]”交换 a[i]和 a[j]。

第 7～13 行的第一轮搜索完成后，将基准数 b 和 a[i]互换，即第 14、15 行。然后，第 16 行“myquicksort(a, h, i－1)”对基准数左边的半个序列进行快速排序；第 17 行“myquicksort(a,i＋1,t)”对基准数右边的半个序列进行快速排序。

第 19 行“a＝[6,10,16,3,8,23,15,7,2,11]”定义列表 a；第 20 行“print(f'Original: {a}.')”输出列表 a。第 21 行“myquicksort(a,0,len(a)－1)”调用快速排序函数对 a 进行排序；第 22 行“print(f'Sorted: {a}.')”输出排序后的列表 a，如图 5-14 所示。

有时在排序的过程中，会同步记录其索引号的变化。这时，需要添加一个索引号的列表，如程序段 5-17 所示。程序段 5-17 在程序段 5-16 的基础上，添加了一个由被排序的列表的元素索引号组成的列表。

程序段 5-17　排序同时更新元素索引号的快速排序法


视频讲解

```
1    def myquicksort(a,p,h,t):
2        if h>=t:
3            return
4        b=a[h]
5        i=h
6        j=t
7        while i!=j:
8            while a[j]>=b and j>i:
9                j-=1
10           while a[i]<=b and i<j:
11               i+=1
12           if j>i:
13               a[i],a[j]=a[j],a[i]
14               p[i],p[j]=p[j],p[i]
15       a[h]=a[i]
16       a[i]=b
17       p[i],p[h]=p[h],p[i]
18       myquicksort(a,p,h,i-1)
19       myquicksort(a,p,i+1,t)
20   if __name__=='__main__':
21       a=[6,10,16,3,8,23,15,7,2,11]
22       p=list(range(len(a)))
23       print(f'Original: {a}.')
24       print(f'Index:    {p}')
25       myquicksort(a,p,0,len(a)-1)
26       print(f'Sorted: {a}.')
27       print(f'Index:    {p}')
```

程序段 5-17 在程序段 5-16 的基础上添加了一个参数 p(第 1 行)，p 表示列表 a 中元素的索引号列表；添加了第 14 行和第 17 行，这两行表示交换列表 a 中的元素时，同步交换元素的索引号；在第 18、19 行中添加了参数 p，即递归调用中需要增加索引号列表 p。

程序段 5-17 的执行结果如图 5-15 所示。由图 5-15 可知，在对列表 a 进行排序的同时，其索引号列表也同步更新了。

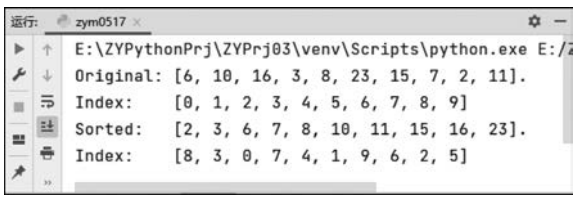

图 5-15　模块 zym0517 执行结果

5.4 复合函数

Python 语言中,有 map 和 filter 函数,可执行类似于数学中复合函数的作用。在前述表 5-1 中曾介绍过这两个函数,这里进一步介绍这两个函数的用法。

map 函数的语法为:

```
map(函数, 可迭代对象 1, 可迭代对象 2, ……, 可迭代对象 n)
```

这里至少要有一个“可迭代对象”,尽可能使全部可迭代对象中的元素个数相同(否则,map 函数将在最短的可迭代对象上停止映射)。map 函数执行的功能为将“函数”作用于“可迭代对象”的每个元素上,如果有多个可迭代对象,则“函数”将平行地作用于多个“可迭代对象”的每个元素上。例如:

```
list(map(lambda x,y,z:x+y+z,[1,2,3],[4,5,6],[7,8,9]))
```

将返回[12,15,18],这里的“lambda x,y,z: x+y+z”为 lambda 函数,也称为匿名函数,语法为:

```
lambda 参数列表(可为空): 表达式
```

例如:

```
f1 = lambda :5 + 3
print(f1())                    #得到 8
f2 = lambda x:x ** 2
print(f2(3))                   #得到 9
f3 = lambda x,y,z:x + y + z
print(f3(1,2,3))               #得到 6
```

在“map(lambda x,y,z: x+y+z,[1,2,3],[4,5,6],[7,8,9])”中 map 函数将 lambda 函数映射到三个列表“[1,2,3],[4,5,6],[7,8,9]”上,x 将依次取列表“[1,2,3]”中的各个元素,y 将依次取列表“[4,5,6]”中的各个元素,z 将依次取列表“[7,8,9]”中的各个元素,然后,根据 x、y 和 z 的取值,依次计算 x+y+z 的值。最后,得到一个 map 对象,转化为列表后为“[12,15,18]”。

filter 函数的语法为:

```
filter(谓词函数, 可迭代对象)
```

所谓的谓词函数是指返回值为逻辑值的函数。filter 函数将“谓词函数”作用于“可迭代对象”的每个元素,返回使得“谓词函数”为真的元素,“过滤”掉那些使“谓词函数”为假的元素。例如:

```
list(filter(lambda x:x>0,[3,10,-5,-7,2,0,12]))      #得到[3,10,2,12]
```

这里 filter 函数将“lambda x: x>0”映射到列表“[3,10,-5,-7,2,0,12]”的每个元素上,“过滤”掉 x 小于或等于 0 的元素,返回一个 filter 对象,转化为列表为“[3,10,2,12]”。

现在比较一下 map 和 filter 函数,例如:

```
list(map(lambda x:x%2==0,[1,2,3,4,5,6,7,8,9,10]))
                        #得到[False,True,False,True,False,True, False, True, False, True]
list(filter(lambda x:x%2==0,[1,2,3,4,5,6,7,8,9,10]))   #得到[2, 4, 6, 8, 10]
```

上述 map 和 filter 均将同一个谓词函数“lambda x：x%2==0”作用于列表“[1，2，3，4，5，6，7，8，9，10]”，根据它们返回的结果可知，map 函数返回“谓词函数”作用于列表中每个元素的结果，而 filter 函数返回列表中使“谓词函数”为真的元素。

5.5　包与模块

Python 语言中有两种包，一种为常规包，另一种为命名空间包。这里仅介绍常规包。在 PyCharm 主界面，选择菜单“文件|新建|Python 软件包”，输入 MyPacks，将在当前项目所在目录下创建一个子目录 MyPacks，其中包括一个文件“__init__.py”（内容为空）。这个子目录 MyPacks 可以称为包。

一般地，可以认为“包”是指包括了很多模块（即文件）的目录（或文件夹），“包”将功能上相关联的一组模块“包”起来。现在在刚刚创建的 MyPacks 包中放入四个模块，模块即文件，分别为 zym0518.py、zym0519.py、zym0520.py 和 zym0521.py。这四个模块的内容分别如程序段 5-18～程序段 5-21 所示。

程序段 5-18　文件 zym0518.py

```
1    def mywelcome(str):
2        print(f'Welcome, {str}.')
3    if __name__ == '__main__':
4        mywelcome('Dr. Zhang')
```

视频讲解

在程序段 5-18 中，第 1、2 行定义了函数 mywelcome。

程序段 5-19　文件 zym0519.py

```
1    import MyPacks.zym0518
2    if __name__ == '__main__':
3        MyPacks.zym0518.mywelcome('Dr. Wang')
```

程序段 5-19 展示了引用其他模块中定义的函数的第一种方法，即第 1 行“import MyPacks.zym0518”，用 import 关键字，其后添加“包名.模块名”。

这种方法要求在引用外部模块中的函数时，要用完整的“包名.模块名”调用其中的函数，例如第 3 行“MyPacks.zym0518.mywelcome('Dr. Wang')”，必须使用形如“包名.模块名.函数名”的形式引用外部模块中的函数。

程序段 5-20　文件 zym0520.py

```
1    import MyPacks.zym0518 as mywel
2    if __name__ == '__main__':
3        mywel.mywelcome('Dr. Wang')
```

程序段 5-20 展示了引用其他模块中定义的函数的第二种方法，即第 1 行“import MyPacks.zym0518 as mywel”，此时，import 关键字给“MyPacks.zym0518”一个别名，即“mywel”，此时可以通过这个“别名.函数名”的形式调用函数。注意：原来的“MyPacks.zym0518”不能再使用。在第 3 行中“mywel.mywelcome('Dr. Wang')”使用形如“别名.函数名”的形式调用外部模块中的函数。

程序段 5-21　文件 zym0521.py

```
1    from MyPacks.zym0518 import mywelcome
2    if __name__ == '__main__':
3        mywelcome('Dr. Wang')
```

程序段 5-21 展示了引用其他模块中定义的函数的第三种方法，即第 1 行“from MyPacks.zym0518 import mywelcome”表示从“MyPacks.zym0518”中装载“mywelcome”，这种方法可以从“包名.模块名”中直接装载其中的“函数名”或“类名”，然后，在当前模块中可以直接使用这些装载的函数或类，如第 3 行“mywelcome('Dr. Wang')”直接调用 mywelcome 函数，就像 mywelcome 函数是本模块中定义的函数一样。

上述介绍的 MyPacks 包和其中四个模块的结构如图 5-16 所示。

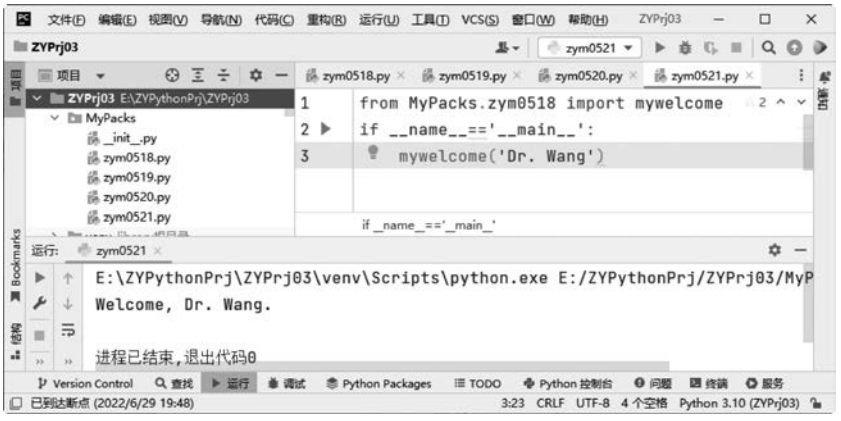

图 5-16 MyPacks 包与其模块的组织结构

上面三种引用外部模块中定义的函数(或类)的方法均为常用方法。此外，同一个工程下的各个模块可以互相引用(需要用 import 装载那个模块名(即文件名))。

Python 语言内置的常用模块有 time、datetime、calendar、os、sys、random、shutil、pathlib、json 和 pickle 等。这里仅介绍 time 和 random 两个包，其余包请参考 Python 在线帮助文档。

time 模块在 4.4 节已作了详细介绍，这里重点介绍借助 time 模块统计程序运行时间的方法，如程序段 5-22 所示。


视频讲解

程序段 5-22 程序运行时间统计实例

```
1    import time
2    if __name__ == '__main__':
3        t1 = time.time()
4        time.sleep(2)
5        t2 = time.time()
6        print(f'The program runs:{t2 - t1:.4f} seconds.')
```

在程序段 5-22 中，第 1 行“import time”装载 time 模块。第 3 行“t1＝time.time()”和第 5 行“t2＝time.time()”记录当前的时间戳，所谓的“时间戳”是指自 1970 年 1 月 1 日 0 时开始按秒计算至当前时间的计数值，第 4 行“time.sleep(2)”表示等待 2s。第 6 行“print(f'The program runs：{t2－t1：.4f} seconds.')”输出程序执行时间。

程序段 5-22 的执行结果如图 5-17 所示。

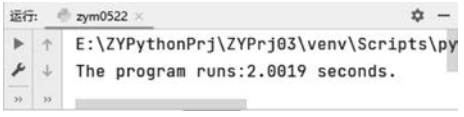

图 5-17 模块 zym0522 执行结果

random 模块用于生成伪随机数，其常用函数如表 5-2 所示。

表 5-2　random 模块常用函数

序　号	函　数　名	典　型　用　法
1	random	random()返回 0～1 的浮点数
2	randint	randint(1,10)返回 1～10(含 10)的整型随机数
3	uniform	uniform(1.0,3.0)返回 1.0～3.0 的均匀分布伪随机数
4	choice	choice([1,2,3,4,5])从列表中随机选取一个元素
5	sample	sample([1,2,3,4,5],3)从列表中随机选取 3 个元素(无放回取样)
6	seed	seed()使用系统时钟作为“种子”；seed(299792458)将“种子”设为 299792458

程序段 5-23 介绍了表 5-2 中各个函数的用法。

程序段 5-23　模块 random 中的函数用法实例


视频讲解

```
1    import random
2    if __name__ == '__main__':
3        random.seed(299792458)
4        print('random():',random.random())
5        print('randint(1,10):',random.randint(1,10))
6        print('uniform(1.0,3.0):',random.uniform(1.0,3.0))
7        print('choice([1,2,3,4,5]):',random.choice([1,2,3,4,5]))
8        print('sample([1,2,3,4,5],3):',random.sample([1,2,3,4,5],3))
9        print('sample([1,2,3,4,5],5):',random.sample([1,2,3,4,5],5))
10       random.seed()
```

在程序段 5-23 中，第 1 行“import random”装载模块 random。第 3 行“random.seed(299792458)”设定伪随机数发生器的种子为“299792458”，这里设定“种子”值的目的在于使得每次执行程序段 5-23 时运行结果相同。第 10 行“random.seed()”将系统时钟值设定为“种子”值，即恢复伪随机数发生器的默认设置，取消第 3 行语句的“种子”配置。

第 4 行“print('random()：',random.random())”输出 random 函数返回的 0～1 的一个伪随机数。第 5 行“print('randint(1,10)：',random.randint(1,10))”输出 randint(1,10)返回的一个 1～10(含)的伪随机整数。第 6 行“print('uniform(1.0,3.0)：',random.uniform(1.0,3.0))”返回 1.0～3.0 均匀分布的一个伪随机实数。第 7 行“print('choice([1,2,3,4,5])：', random.choice([1,2,3,4,5]))”输出随机从列表[1,2,3,4,5]中选出的一个数值。第 8 行“print('sample([1,2,3,4,5],3)：',random.sample([1,2,3,4,5],3))”输出随机从列表[1,2,3,4,5]中选出的三个数值(无放回取样)。第 9 行“print('sample([1,2,3,4,5],5)：',random.sample([1,2,3,4,5],5))”将列表[1,2,3,4,5]中的元素随机排列，这是因为 sample 使用无放回取样，列表的长度为 5，(无放回)随机取样的次数也是 5，所以，这里实际上是随机排列列表[1,2,3,4,5]。

程序段 5-23 的执行结果如图 5-18 所示。

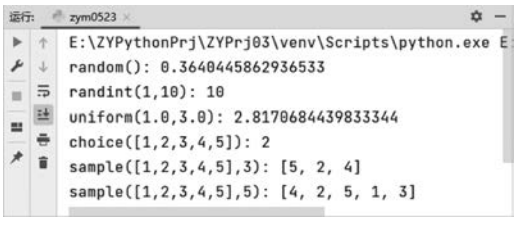

图 5-18　模块 zym0523 执行结果

5.6 本章小结

函数在各种计算机语言中均起到不可或缺的作用,即使在汇编语言中也需要编写大量的“函数”。函数的主要作用为增强程序代码的复用性和可读性,从而可以借助函数编写大规模的程序。本章详细介绍了 Python 语言的常用内置函数,这些内置函数需要熟练掌握。然后,介绍了设计自定义函数的方法,重点介绍了函数定义与调用方法、可变参数函数设计方法、函数返回值与变量作用域、函数闭包与装饰器以及递归函数等,接着,介绍了复合函数的用法。最后讨论了包与模块的定义与调用方法。

在学习了函数之后,Python 语言的基础语法部分只剩下“类”这一概念了,类是封装了数据(称为属性)和作用于这些属性上的方法(即函数)的一种“类型”。第 6 章将全面介绍类的定义与使用方法。

习题

1. 定义一个函数 rect,形式参数为长方形的宽 width 与高 height,返回长方形的面积 area。编写主程序,输入长方形的宽和高,然后,调用 rect 函数计算并输出长方形的面积。

2. 借助函数编写程序,输入两个正整数,求这两个正整数的最大公约数和最小公倍数。

3. 编写函数,实现两个整数、两个双精度浮点数和两个复数的乘法运算。

4. 编写程序,从键盘上输入 10 个整数,然后,对输入的整数序列进行降序排列,并输出排列好的整数序列(使用冒泡法、选择法和快速排序法三种方法实现)。

5. 编写函数,实现两个二维整型数组的乘法运算。一个 m 行 n 列的矩阵 $\boldsymbol{A}$ 乘以一个 n 行 k 列的矩阵 $\boldsymbol{B}$ 的积为一个 m 行 k 列的矩阵 $\boldsymbol{R}$:

$$\boldsymbol{R}_{i,j}=\sum_{t=1}^{n}\boldsymbol{A}_{i,t}\boldsymbol{B}_{t,j},\quad i=1,2,\cdots,m,\quad j=1,2,\cdots,k$$

其中,$\boldsymbol{R}_{i,j}$ 表示 $\boldsymbol{R}$ 矩阵的第 i 行第 j 列的元素。

6. 编写递归函数计算 x^n,其中 n 为整数,x 为实数。

7. 使用与程序段 5-15 不同的递归方法计算汉诺塔问题。提示:将汉诺塔问题分解为①终止条件:当 A 柱上只有一个盘子时,将该盘子从 A 柱移动到 C 柱。②递归调用:将 A 柱上的 $n-1$ 个盘子借助 C 柱移动到 B 柱;将 A 柱上的第 n 个盘子(最大的盘子)从 A 柱直接移动到 C 柱;将 B 柱上的 $n-1$ 个盘子借助 C 柱移到 A 柱上。这样将 A 柱中最大的盘子转移到 C 柱了,并且 A 柱上剩下 $n-1$ 个盘子。注意:该方法得到的结果会比程序段 5-15 得到的结果复杂一些,因为这种方法实际上将 n 个盘子的汉诺塔问题转化为三个 $n-1$ 个盘子的汉诺塔问题。

8. 在快速排序法中,选择基准数为列表中间的元素,按照快速排序法的原理,编写函数实现这种形式的快速排序。