

# 第一章 快速入门

## 本章导读

Python(原意:大蟒蛇),是一种面向对象的解释型程序设计语言,由Guido van Rossum(吉多·范·罗苏姆)于1989年底发明。1991年,Python第一个公开版发行。Python源代码遵循 GPL(通用公众许可,General Public License的简写,即许可用户运行和复制软件自由,发行传播软件自由,获得软件源码自由,更改软件并将更改软件发行传播自由,通常GPL软件为免费软件)。

Python语法简洁而清晰,具有丰富和强大的类库。它常被昵称为**胶水语言(Glue Language)**,能够把用其他语言制作的各种模块(尤其是C/C++)很轻松地联结在一起。常见的一种应用情形是,使用Python快速生成程序的原型,然后对其中有特别要求的部分,用更合适的语言改写,比如3D游戏中的图形渲染模块,性能要求特别高,就可以用C/C++重写,而后封装为Python可以调用的扩展类库。

Google公司内部很多项目,例如Google Engine使用C++编写性能要求极高的部分,然后用Python或Java/Go调用相应的模块,他们的口号是:“Python where we can, C++ where we must(除非必须C++,尽量用Python)。”

## 学习目标:

1. 了解Python安装及其运行环境;
2. 掌握Python变量、数据类型、运算符以及表达式等等;
3. 掌握Python流程控制语句;
4. 掌握Python序列数据类型如str、list、tuple、set以及dict等;
5. 掌握Python切片操作;
6. 掌握基本文件操作;
7. 基本掌握功能扩展库的引入以及第三方库的安装;
8. 初步掌握函数定义以及lambda函数;
9. 掌握列表解析概念及其使用。

## 本章目录

### 第一节 初识Python

- 1、安装Python系统
- 2、初识Python程序
- 3、变量与数据类型
- 4、常用函数
- 5、运算符
- 6、功能扩展库

### 第二节 控制语句

- 1、if-else选择语句
- 2、for-in循环
- 3、while循环

### 第三节 Python的缩进

### 第四节 函数定义

### 第五节 lambda函数

### 第六节 序列类型

- 1、Str(字符序列)
- 2、list(列表序列)
- 3、Tuple(元组序列)
- 4、Set(集合类型)
- 5、Dict(字典序列)

### 第七节 序列函数

- 1、统计函数: max()、min()和sum()
- 2、过滤函数: filter()
- 3、映射函数: map()
- 4、排序函数: sorted()
- 5、组合函数: zip()
- 6、反序函数: reversed()

### 第八节 列表解析

Python有如下特点：

- Python是解释型脚本语言；
- Python是动态型语言；
- Python是面向对象的语言；
- Python语法简洁；
- Python扩展库丰富；
- Python可编写桌面程序，也可编写服务器端程序；
- Python可编写窗口程序，也可以编写字符界面程序。
- Python是自由软件，源代码和解释器CPython遵循GPL协议。

## 现在，开始Python学习之旅！

### 第一节 初识Python

#### 1、安装Python系统

Python可以安装在各种操作系统平台，如：Windows、Mac、linux等，可以在[www.python.org](https://www.python.org)网站下载对应操作系统的安装文件并安装。如图1-1所示，首先在<http://www.python.org/downloads>下单击鼠标所指处下载Python对应安装程序。Python网站能自动判断下载时所用操作系统及其相应下载。如果要用在其他系统，可选择如图底部的“Release version”对应版本，进入相关网页后下载对应版本即可。此处以Windows安装为例。

Python主要有版本2和版本3两大系列，两者基本原理一致，但用法有不少出入。强烈建议使用Python 3.X系列。本教程以Python3.6及其以上版本为例。

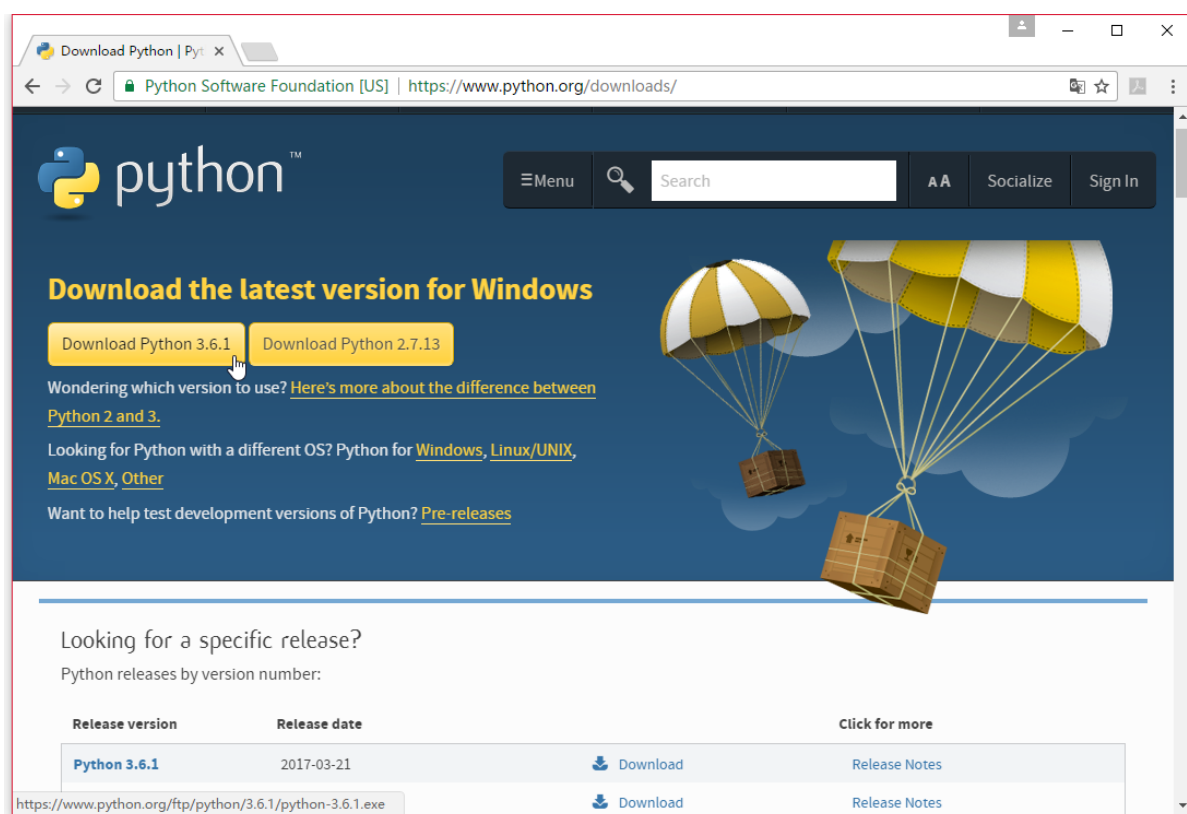


图1-1 Python网站下载Python

运行下载后的Python安装程序，将进入如下图所示的界面。注意：请勾选底部的“Add Python 3.6 to path”以确保Python指令在任何位置都能被执行。建议选择Customize installation(个性化安装)。

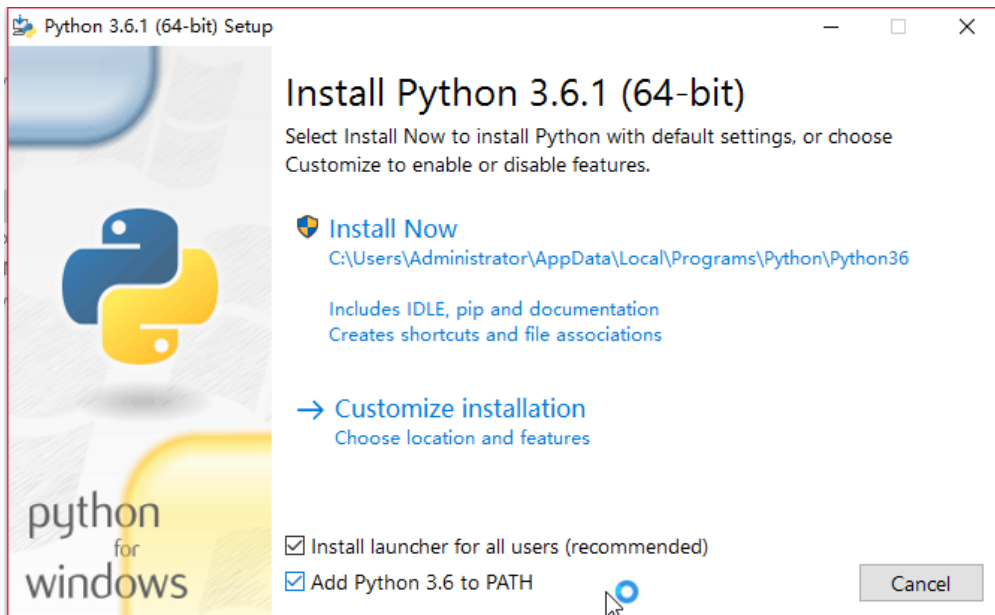


图1-2 Windows环境安装Python(1)

点击 “Customize installation” 后，进入如下界面，建议全部勾选，单击 “Next” 命令按钮，进入下一个界面。

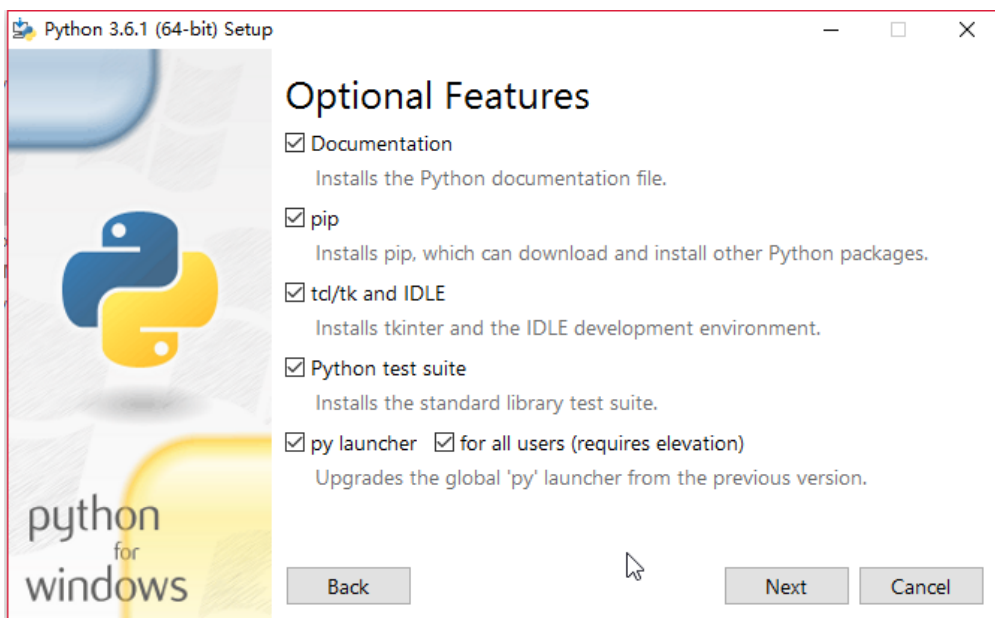


图1-3 Windows环境安装Python(2)

进入最后一个界面，建议按照如图所示选择。另外，如果C盘空间有限，不建议安装在C盘，可以指定安装在D盘或其他盘符，或如图所示的D盘Python36文件夹下。设置完上述选择后，单击 “Install” 按钮即可自动完成安装。

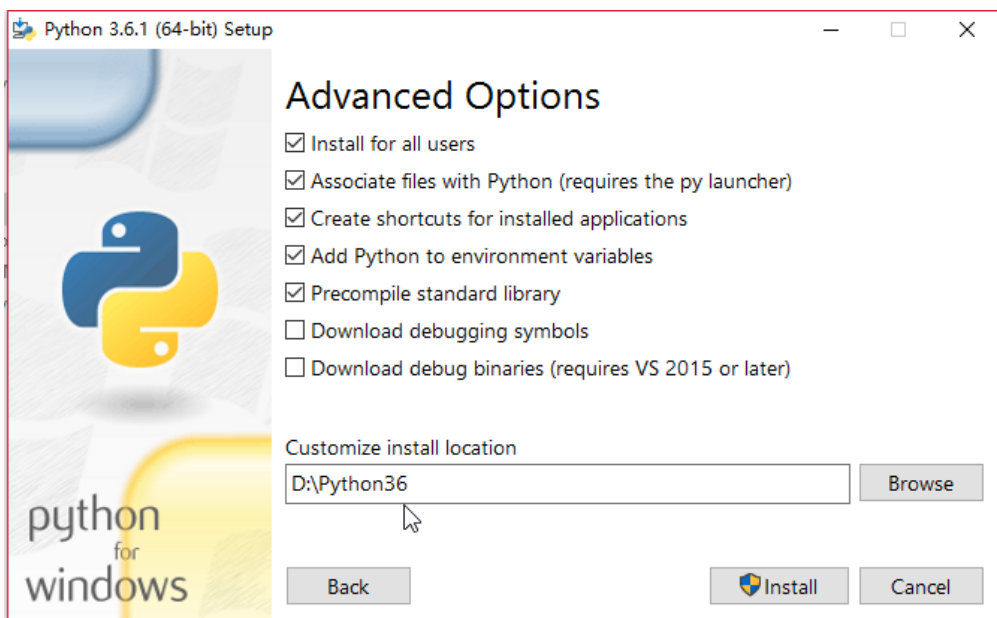


图1-4 Windows环境安装Python(3)

Python安装完毕后，进入Windows的“命令提示符”状态，在Windows10可以右键单击Windows开始图标或搜索“命令提示符”，进入如下图所示的界面。在命令行键入“python -V”（注意是大写V），如果能输出类似Python3.6.1文字，即表明Python安装成功。

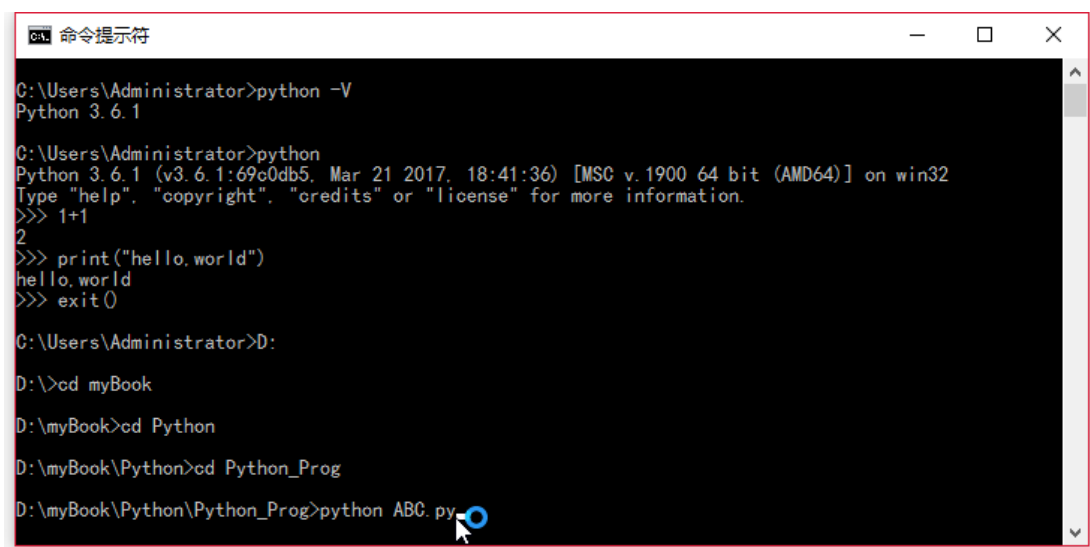


图1-5 Python交互式执行环境

在命令行输入python并回车后，可以进入Python交互式环境，此时会显示版本等信息，其后是>>>提示符(如图1-5所示)，此时可以键入Python指令，如：1+1然后回车，即可得到1+1的结果。与之类似，如输入print("Hello,Python!")则执行结果是输出“Hello,Python!”。Python交互式环境有利于练习各种Python指令，输入命令回车即可得到结果。退出输入exit()函数回车即可。

Python也可以执行Python程序，如图1-5底部的“python abc.py”，其中abc.py即为Python程序文件。程序源代码通常由多行组成。可以进入Python程序所在文件夹执行，也可以在其他位置执行，如：python d:\myBook\Python\Python\_Prog\abc.py。

图1-5中命令行执行“D:”表示改变盘符到D盘，cd(change directory的简写)表示改变文件夹。cd myBook表示进入myBook文件夹，其他与之类似。另外，dir /p能列示文件夹下所有文件和子文件夹。这些命令称之为DOS命令。在Linux或Mac系统中，也有类似功能的命令。不过Linux和Mac没有盘符概念。

## 了解DOS

DOS是Disk Operation System(磁盘操作系统)首字母简写。当前流行的操作系统主要有Windows系列、macOS系列(用于苹果计算机系统)、iOS(用于苹果手机)、Android(Google手机操作系统)以及Linux(可免费使用的操作系统)。这些操作系统是图形化操作系统，用鼠标或手势(gesture)就能方便使用。在图形化操作系统之前，主要是字符界面操作系统，如DOS等。使

用时需要在命令行输入各种命令，记忆命令也就成为必须，也因此成为计算机普及的难题。随着图形化操作系统推出，难题迎刃而解，计算机普及成为必然。

在D:\myBook\Python中，D:表示盘符，也就是常说的D盘。在Windows系列计算机中，一般都有C盘和D盘，但并不意味着必然有两块硬盘，通常都是将一块物理硬盘分成两个或多个逻辑区，然后给每个逻辑区命名。C和D就是其中两个逻辑区的名字。注意：在Mac系列计算机中，没有盘符概念。

在D:\myBook\Python中，myBook和Python称之为文件夹或者目录，Python为myBook的子文件夹。在DOS中，用MD命令建立新文件夹。RD删除已经存在的文件夹，删除时必须确保该文件夹下没有子文件夹和文件。DIR能列出当前文件夹下的文件和文件夹。

COPY也是DOS常用命令，用于复制文件，其格式为**COPY 起源 目标**，如copy abc.txt xyz.txt(复制当前文件夹下的abc.txt为xyz.txt)。复制时还可以指定文件夹，如copy D:\myBook\pkuLogo.gif pku.gif(将D盘myBook文件夹下的pkuLogo.gif文件复制到当前位置且命名为pku.gif)。如果省略“目标”，则是复制“起源”到当前位置且文件名保持不变。

dir \*.gif能列出当前文件夹下所有扩展名为gif的文件，其中\*被称之为通配符，代表所有0或者多个字符，\*.gif匹配所有扩展名为gif的文件，dir \*.gif则是列出所有扩展名为gif的文件。dir i\*.html则是列出所有以i开始扩展名为html的文件。在DOS，?也是通配符，代表0或者1个字符，比如：dir i?.html则代表以i开始，其后跟0个或1个字符的文件，比如：i.html、i1.html、in.html等等。

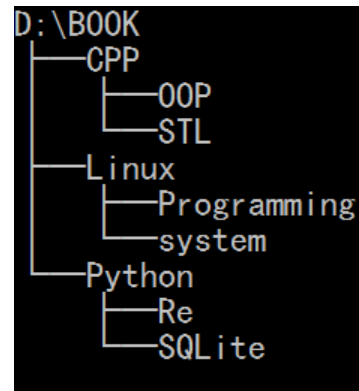


图1-6 树形文件夹结构

在DOS，DEL用于删除文件，如del abc.jpg能删除当前文件夹下的abc.jpg；del \*.html则删除当前文件夹下的所有html文件。

几乎所有的操作系统都是用树形结构组织文件夹和文件。左图是DOS的tree book命令的执行结果，其中tree是DOS命令，book是指定文件夹的名称，如果省略则是显示当前文件夹的树形结构。假定当前位置在SQLite。如果要列出Re文件夹下的内容，则可以执行dir ../Re，其中..代表当前文件夹的上级文件夹，在此处是指Python；dir ../..则可以列出Book文件夹的内容。在DOS中用..代表上级文件夹，.代表当前文件夹，如dir .则是列出当前文件夹下的内容，当然此时一般省略。

在DOS或很多其他操作系统中，通常还用\代表根文件夹，如dir \则是列出D盘下第一层文件夹和文件(假设当前在D盘任意位置)。dir c:\则是列出C盘根文件夹下的内容；dir c:\xyz则是列出C盘根文件夹下xyz文件夹的内容。

另外，DOS和Windows不区分文件夹或文件名称中英文字母的大小写，但苹果系统或者其他Unix或Unix-like(Unix系统或类Unix系统)都区分大小写即大小写敏感。

2、初识Python程序

交互式环境利于执行短小命令，熟悉新的语法特征，但不能保存执行过的所有代码。Python代码可以撰写在一个文件之中，然后执行。如例程1-1所示。

例程1-1

```
第1行 | #第一个Python程序
第2行 |
第3行 | print("Hello,Python!") #输出: Hello,Python!
第4行 |
第5行 | #eof
```

编辑完上述代码后，保存为Hello.py文件，编码格式为UTF-8。在Windows环境下，可以如图1-6所示执行。

在linux或者其他Unix-like环境，可在代码前指定Python解释器位置，然后设置文件为可执行，即可直接运行，如例程1-2所示。该代码被保存在名为hello.py文件中，在Unix-like环境下，执行chmod a+x hello.py，然后直接执行该文件即可。如输入：./hello.py等。

例程1-2

```
第1行 | #! /usr/bin/python
第2行 |
第3行 | print("Hello,Python!") #输出: Hello,Python!
```



第4行  
第5行

#eof



图1-7 Python交互式执行环境

例程1-2和例程1-3都有符号#出现，该符号后的内容为注释。注释对程序执行效果没有影响，但有利于代码相关人更好地阅读和理解程序功能。

例程1-3第4行代码a=5的功能是让a的值为5，这种语句称之为赋值语句，a称之为变量，第5行与之类似。第7行c=a+b的含义是让a和b执行加法运算，并将结果赋值给c。符号+以及符号=称之为运算符。很多数学运算符大都可以在程序设计语言中使用，其含义也基本相同。如\*表示乘法、/表示除法、+表示加法、-表示减法、//表示整除、%表示整数除法求余数、\*\*表示乘方等等。由运算符连接的式子称之为表达式，如a+b是表达式，a\*b也是表达式。

例程1-3

第1行  
第2行  
第3行  
第4行  
第5行  
第6行  
第7行  
第8行  
第9行  
第10行  
第11行  
第12行  
第13行  
第14行  
第15行  
第16行  
第17行

```
#!/usr/bin/python
##符号后的内容为注释，对程序执行效果没有影响。
a=5 #赋值语句
b=3
c=a+b #+以及类似的符号被称之为运算符
print(c) #print及其括号被称为函数，c称之为函数的参数
#print()的功能是输出括号中参数的值
print(a**b) #输出：125
print(b/a) #输出：0.6
print(b//a) #输出：0
print(a%b) #输出：2
#eof#eof的含义是文件结束，用于标记程序结束是一个好习惯
```

例程1-3第9行的print(c)中的print()称之为函数，print是函数名。函数的英文是function，与数学中f(x)有相同之处。f(x)中的x称为变量或者参数，print(c)中的c在程序设计中也称为参数。例程中print()的功能是输出参数的结果。不同的函数有不同的功能，如sin()和cos()分别是正弦函数和余弦函数，功能不同，计算机语言与之类似。

上述例程中的每个语句按出现的先后将被依次执行，但例程1-4第6行能否被执行受到第5行控制。如果a是偶数则将被执行，如果a是奇数则将不被执行。这种语句称之为选择语句(Choice Statement)。根据Python规定，第6行必须用空格或Tab键缩进，否则将导致错误。另外，第5行最后的冒号也是必须，不能省略。第10-13行也是选择语句，多了if语句的else部分，其含义是如果a%3的结果大于0则执行第11行否则执行第13行。

例程1-4

第1行	#!/usr/bin/python
第2行	
第3行	a=1188
第4行	
第5行	if a%2==0: #英文冒号不能省略
第6行	print("a是偶数! ")
第7行	
第8行	#上述语句执行完毕输出: a是偶数!
第9行	
第10行	if a%3>0:
第11行	print("不能被3整除! ")
第12行	else: #英文冒号不能省略
第13行	print("能被3整除! ")
第14行	
第15行	#上述语句执行完毕输出: 能被3整除!
第16行	
第17行	#eof

第5行代码中的==以及第10行代码中>也是运算符，称之为关系运算符，其他关系运算符还有>=(大于等于)、<(小于)、<=(小于等于)、!=(不等于)、<>(不等于)，与数学中的运算符定义基本相同。关系运算符的结果只有两种可能，分别是True和False(注意：首字母大写)。要注意区分=和==，前者是赋值运算符，将其右边结果放到左边变量中，而后者==是关系运算符用于判断左右两侧是否相等。

在计算程序设计语言中，顺序语句、选择语句和循环语句(Loop Statement)构成了代码的基本语句，或者说程序设计主要就是调度这3种语句。循环语句将在后续小节讲解。

### 3、变量与数据类型

计算机编程语言的变量(英文：variable)一词源于数学，用于保存初识值或者程序计算结果，变量值可通过变量名访问。和很多语言一样，Python语言的变量也必须先声明后使用，如例程1-5所示。

例程1-5

第1行	#!/usr/bin/python
第2行	
第3行	#print(a)#错误! 没有定义
第4行	
第5行	a=123
第6行	print(a)#输出: 123
第7行	
第8行	a="ABCD"
第9行	print(a)#输出: ABCD
第10行	
第11行	#eof

在上例中，第3行代码如果将第1个注释符号删除，则将导致错误，Python提示为“NameError: name 'a' is not defined”(名称错误：名称a没有定义)。第5行代码a=123中的a为变量名称，其值为123。和有些语言不同，在Python中申明变量无需指定数据类型，其类型由Python自动推断，此处为int(整数)。在第8行中，变量a被赋值为"ABCD"，其类型为str(字符串，由一个或多个字符组成)。和有些语言不同，在Python中，同一个变量名称可以先后被赋值为不同类型。

type()是一个Python函数，用于返回变量的数据类型，如例程1-6所示，其中class表示类，是面向对象体系的重要概念(将在后续章节讲解)。从该例程第4行可以看出，此时valA的类型为int类，即整数类，然后分别是str类、bool类。

例程1-6

```

第1行  #变量与数据类型
第2行
第3行  valA=100
第4行  print(type(valA))#输出<class 'int'>
第5行
第6行  valA="ABC" #valA此时为字符串，两侧有引号包围，单双引号均可，常用于保存字符中文等
第7行  print(type(valA))#输出<class 'str'>
第8行
第9行  valA=True
第10行 print(type(valA))#输出<class 'bool'>
第11行
第12行  #eof

```

在Python中，可以一次给多个变量赋值，也可以将一个值付给多个变量，如例程1-7所示。

#### 例程1-7

```

第1行  #!/usr/bin/python
第2行
第3行  a,b,c=4,5,"ABC"
第4行  print(a,b,c)#输出： 4 5 ABC
第5行  a=b=c=99
第6行  print(a,b,c)#输出： 99 99 99
第7行
第8行  a=b=c="Hello!Python!!!"
第9行  print(a,b,c)#输出： Hello!Python!!! Hello!Python!!! Hello!Python!!!
第10行
第11行  c="Python"
第12行
第13行  #eof

```

在Python中，必须相同的类型才能在一起运算，如例程1-8第7行所示即为错误，Python错误提示为：TypeError: unsupported operand type(s) for +: 'int' and 'str'(类型错误：对加号不支持int与str运算)。第9行实现了a与b相加，其原因是代码int(b)将b转换为int类型。

#### 例程1-8

```

第1行  #!/usr/bin/python
第2行
第3行  #print(a)#错误！没有定义
第4行
第5行  a=123
第6行  b="345"
第7行  #print(a+b) #错误！不同类型不能直接在一起运算
第8行
第9行  print(a+int(b)) #通过int()将b转换为int类型，此处输出： 468
第10行
第11行  #eof

```

Python语言中，基本数据类型有int(整数类)、float(浮点类，能带小数点的数)、bool(布尔类)与str(字符串类)。除str外，其余三种属于Numbers类型(可以理解为数值类型)，之间能互相运算。而str类与其混合运算时，必须进行类型转换。bool类型只有两种



结果True和False。当将其他类型转换为bool型时，对于数值，如果不为0，则转换为True(第19行)，否则转换为False(第20行)；对于字符串，空字符串转换为False(第22行)，其他为True(第23行)。例程1-9是类型转换示例。

#### 例程1-9

```
第1行  #! /usr/bin/python
第2行
第3行  a=11
第4行  b=34.56
第5行  print(a+b) #输出： 45.56
第6行
第7行  print(11+True)#True的值为1，此处输出： 12
第8行
第9行  print(11+False)#False的值为0，此处输出： 11
第10行
第11行 print(a+int(b))#通过int()将b转换为int类型，此处输出： 45
第12行 print(float(a)+b)#输出： 45.56
第13行
第14行 c="43.21"
第15行
第16行 print(float(c)+a)#输出： 54.21
第17行 print(int(float(c))+a)#直接写int(c)将导致错误，输出： 54.21
第18行
第19行 print(bool(12))#输出： True
第20行 print(bool(0))#输出： False
第21行
第22行 print(bool(""))#输出： False
第23行 print(bool("A"))#输出： True
第24行
第25行 #eof
```

Python还提供序列数据类型，使得一个变量可以容纳多个数据，如例程1-10第3行所示，多个数值用逗号分开，并界定在配对的方括号之中，这种类型被称为list(列表)类型(注：还有其他序列类型)，可以将多个值甚至不同类型的值放在方括号之中，然后通过下标(第11行)访问其中的成员。对于list，还可以用切片(slice)操作，如第12行a[0:3]所示，即截取编号为0到编号3(不含)的部分成员，截取出的片段也是list类型，如同子集操作。例程1-10的操作对所有Python序列类型都适用，Python的str同样序列类型，同样适用。

#### 例程1-10

```
第1行  #! /usr/bin/python
第2行
第3行  a=[12,23,7,34,56,8,10,7]
第4行  b=a+[99,100,101]#+运算符功能是将两个list拼接，a不发生改变
第5行  c=[1,2,3]
第6行  print(b)#输出： [12, 23, 7, 34, 56, 8, 10, 7, 99, 100, 101]
第7行  print(c*3)#输出： [1, 2, 3, 1, 2, 3, 1, 2, 3], *运算符实现指定次数重复
第8行
第9行  print(a)#输出： [12, 23, 7, 34, 56, 8, 10, 7]
第10行
第11行 print(a[0])#下标访问，输出： 12
```

```

第12行 print(a[0:3])#切片操作, 输出: [12,23,7]
第13行
第14行 print(len(a))#len()返回序列a的长度(数据个数)
第15行 print(min(a))#min()返回序列a的最小值
第16行 print(max(a))#max()返回序列a的最大值
第17行 print(7 in a)#输出: True, in是运算符, 判断是否在其中
第18行 print(7 not in a)#输出: False, not in是运算符, 判断是否不在其中
第19行
第20行 print(a.count(7))#count()用于返回出现次数, 此处输出2
第21行 print(a.index(7))#index()用于返回7第1次出现位置, 此处输出2
第22行 print(a.index(7,3))#index()用于返回7在编号为3的位置及其后第1次出现位置, 此处输出7
第23行 print(a.index(7,1,5))
第24行 #index()用于返回7在编号为1的位置及其后第5号位置之前第1次出现位置, 此处输出2
第25行
第26行 #eof

```

例程1-11是上例中count()在str字符串中的应用示例, 统计字符“秋”出现的次数。

#### 例程1-11

```

第1行 #!/usr/bin/python
第2行
第3行 #统计林黛玉之秋窗风雨夕
第4行
第5行 linQiu=""
第6行 秋花惨淡秋草黄, 耿耿秋灯秋夜长。
第7行 已觉秋窗秋不尽, 那堪风雨助凄凉!
第8行 助秋风雨来何速? 惊破秋窗秋梦绿。
第9行 抱得秋情不忍眠, 自向秋屏移泪烛。
第10行 泪烛摇摇爇短檠, 牵愁照恨动离情。
第11行 谁家秋院无风入? 何处秋窗无雨声?
第12行 罗衾不奈秋风力, 残漏声催秋雨急。
第13行 连宵脉脉复飏飏, 灯前似伴离人泣。
第14行 寒烟小院转萧条, 疏竹虚窗时滴沥。
第15行 不知风雨几时休, 已教泪洒窗纱湿。
第16行 """
第17行
第18行 qiuCount=linQiu.count("秋")
第19行 print(qiuCount)#输出: 15
第20行
第21行 #eof

```

除了list类型外, Range也是常用的序列类型, 用于产生规则变化的整数序列, 如例程1-12所示。Range类型可以转换为list类型, 如第11-12行所示。

#### 例程1-12

```

第1行 #!/usr/bin/python
第2行
第3行 #range()的格式: range(start,end,step), 不包含end值, step默认为1

```

```

第4行
第5行 rangeA=range(5,10)#相当于[5,6,7,8,9], 不包含10, step默认为1
第6行 rangeB=range(10)#相当于[0,1,2,3,4,5,6,7,8,9], 不包含10
第7行 rangeC=range(5,10,2)#相当于[5,7,9], 不包含10
第8行 rangeD=range(0,-10,-2)#相当于[0,-2,-4,-6,-8], 不包含-10
第9行 rangeE=range(0,-10)#相当于[], 空没有值
第10行
第11行 print(range(10)[3:6])#range切片, 输出: range(3,6)
第12行
第13行 listA=list(range(100,200))#产生100到200之间的整数序列, 很方便
第14行 print(type(listA))#输出: <class 'list'>
第15行
第16行 #eof

```

序列类型, 除了list、range和str外, 还有dict、tuple、set等类型, 请查看本章其他章节。

#### 4、常用函数

##### (1) input函数

在Python中, input()用于键盘输入, 如例程1-13所示。

例程1-13

```

第1行 #input()函数的使用
第2行
第3行 input("请输入您的姓名: ")#仅仅输入而已
第4行 myName=input("请输入您的姓名: ")#将input()函数输入内容放入变量myName之中
第5行 print(myName)#将myName中的内容用print()函数输出
第6行
第7行 #eof

```

例程1-14用input()函数输入数值, 但当与数值相加时发生错误, **其原因是input()函数将输入都作为字符串**, 如果需要转换数值, 可以考虑用int()函数将之转换为整数, 用float()函数将之转换为浮点数。

例程1-14

```

第1行 #input()函数的使用
第2行
第3行 numA=input('请输入第一个数: ')#假如输入:100
第4行 numB=input('请输入第二个数: ')#假如输入:10
第5行
第6行 #print(numA+100), 如此书写将导致错误
第7行 print(numA+numB)#计算出numA+numB的值, 输出为10010
第8行 print(int(numA)+int(numB))#输出为: 110
第9行
第10行 numC=float(input('请输入第一个数: '))#假如输入:100
第11行 numD=float(input('请输入第二个数: '))#假如输入:10
第12行 print(numC+numD)#计算出numA+numB的值, 输出为110.0
第13行
第14行 #eof

```

##### (2) print函数

例程1-1中的print()是最为常用的输出函数，在例程1-15中print()函数有了更多用法。

#### 例程1-15

```
第1行 #print()函数的使用
第2行
第3行 print("Hello,Python!") #输出: Hello,Python!
第4行 print("Hello","Python!")#可以输出多个值, 输出: Hello Python!
第5行 print(100+200)#可以直接输出表达式的结果, 输出: 300
第6行 print("100+200=",100+200)#输出: 100+200=300
第7行
第8行 #eof
```

在Python中，print()使用相当频繁，为更好地控制输出，print()还提供不少选项，sep参数用于设定每个输出项之间的间隔符，end设定输出每个print()函数执行完毕后紧跟字符，默认为回车换行。end和sep在函数中被称之为命名参数(将在后续章节讲解)，如果end和sep同时使用，二者没有先后顺序，其使用方法如例程1-16所示。

#### 例程1-16

```
第1行 #print()函数的多种选项
第2行
第3行 print(12,34,13.12,sep='$$$')#设定输出项之间的间隔符, 输出: 12$$$34$$$13.12
第4行
第5行 print(12)#输出12, 然后换行
第6行 print(24)#输出24, 然后换行
第7行 print(12,end=" ")#输出12, 不换行, 输出空格
第8行 print(24)#输出24, 然后换行。即24和上行的12在同一行显示
第9行
第10行 strA="ABCXYZ"
第11行 print("字符串%s的长度为%d"%(strA,len(strA)))#输出: 字符串ABCXYZ的长度为6
第12行 #注意最后一个双引号后的%不能省略
第13行
第14行 #eof
```

print()函数常和转义字符(escaped character)一起使用，如例程1-17所示，其执行效果如图1-6所示。第3行的输出效果如图前3行所示。之所以如此输出，是因为Python将\n解释成为换行而不是两个普通字符，因此称之为转义字符。Python的转义字符与C/C++基本相同。在转义字符中，\t输出横向制表位，一般是8个字符，因此“曹雪芹”前有8个空格(一个汉字占两个英文字符宽度)。如果要输出\本身，则可以用\\表示，输出单引号双引号与之类似，如例程第6行所示。

第7行是\t的应用示例，每个制表位占8个英文字符宽度。当不足8个字符时，字符后紧接余下空格，“12”后面空出相当于6个空格，“123456”后紧跟2个空格。“1234567”后则紧跟1个空格。如果是“12345678”后紧跟\t，则该制表位占满，将空出8个空格位置。

第8行代码与第3行几乎相同，唯在字符串引号前多了一个r(raw的简写)，其含义是禁止转义，因此第9行输出时原样输出，没有发生转义。

当每次执行print()函数时，默认将\n作为最后输出字符，可以通过关键字end设定print()函数执行后的输出字符，如例程第12-13行所示以\t作为最后输出字符。如果将\t改成1个空格，则每个数字后紧跟一个空格。如果是空字符串即字符串引号对界定范围内没有任何字符，则所有输出之间连续没有间隔。

#### 例程1-17

```
第1行 #转义字符的使用
第2行
第3行 strA="满纸荒唐言，一把辛酸泪！\n都云作者痴，谁解其中味？\n\t曹雪芹"
第4行
```

```

第5行 print(strA)
第6行 print("\\t\\t\\t") #如果要输入斜杠，可以使用双斜杠
第7行 print("\\t输出:\\t12\\t123456\\t1234567\\t1234567\\tABC")
第8行
第9行 strA=r"满纸荒唐言，一把辛酸泪！\\n都云作者痴，谁解其中味？\\n\\t曹雪芹"
第10行 print(strA)
第11行
第12行 for i in range(10):
第13行     print(i,end= "\\t")
第14行
第15行 #eof

```

```

满纸荒唐言，一把辛酸泪！
都云作者痴，谁解其中味？
曹雪芹
\\t输出: 12      123456  1234567 1234567 ABC
满纸荒唐言，一把辛酸泪！\\n都云作者痴，谁解其中味？\\n\\t曹雪芹
0      1      2      3      4      5      6      7      8      9

```

图1-8 例程1-17执行效果

print()函数尚有更多格式选项，如第11行中的%s与%d表示其所在位置由其后%后括号中相应项目替换(更多说明请参考：<http://www.cnblogs.com/graceting/p/3875438.html>)。另外，print()还可以直接输出到文件，请参看本章“文件操作”部分。

### (3) format函数

format函数更确切的说法是字符串对象的方法。方法是对象中的函数，如list.count()中，list代表list对象，count()是list对象的函数。字符串对象在Python是str，因此format函数的正确写法是str.format()，其中str可以是各种字符串，format()中有多种参数，易于格式化输出，如例程1-18所示。

总体来讲，format()函数以英文句点附在str(字符串)对象之后，如例程第3行所示，其中花括号{}将是被替换内容，如第3行两个花括号依次被format()函数的两个参数替换。花括号中有各种变化，总体来讲，分为两个部分，分别以冒号分隔。冒号前是序号或名称，冒号后是格式。如例程第4行，{0:3.2f}中第1个0是format()参数中编号为0的参数即第1个参数，冒号后的3.2f是定义格式，此处f代表float即带小数点的数，3.2代表总长度3，其中小数点后保留两位。{1:e}表示第1个参数，e表示按科学计数法输出。

第5行代码中的{0}和{1}分别代表编号为0和1的参数，可以重复引用。对于命名参数，可以直接使用关键字名称，如第7行所示。第11行代码{0[0]}中的第1个0如前所述，代表编号为0的参数，在此处概述为list对象，[0]是该对象的下标运算。

第14-17行表示对齐等。如代码("{!^8}"中冒号前没有数字，表示按序参数，此处仅有1个参数，就代表该数。!^8中的8代表输出时占8个字符，^代表居中对齐，!可以换成其他符号如第15行的0等表示填充。即总长度8个字符，如果不到8个字符两侧用!填充。与^对等的还有<左对齐如第17行，>右对齐如第14-15行。

第23行冒号后的逗号表示按千分位隔断数字。第25行的bdox分别表示二进制、十进制、八进制和十六进制吗，即输出时可以将数值转换成相应进制。

### 例程1-18

```

第1行 #format使用示例
第2行
第3行 print("{}{}".format("李白","杜甫")) #输出: 李白,杜甫
第4行 print("{0:3.2f},{1:e}".format(3.1415926,2.718281828459)) #输出: 3.14,2.718282e+00
第5行 print("{1},{0},{1}".format("李白","杜甫")) #输出: 杜甫,李白,杜甫
第6行
第7行 print("{LiBai},{DuFu}".format(LiBai=701,DuFu=712))#输出: 701,712
第8行 #LiBai,DuFu在此处是命名参数

```

```

第9行
第10行 poetList=["李白","杜甫","王维","王昌龄","岑参"]
第11行 print("{0[0]},{0[2]},{0[1]},{0[3]}".format(poetList))
第12行 #输出: 李白,王维,杜甫,王昌龄
第13行
第14行 print("{:>8}".format(1640))#输出: 1640
第15行 print("{:0>8}".format(1640))#输出: 00001640
第16行 print("{!^8}".format(1640),"年, 明亡!")#输出: !!1640!!年, 明亡!
第17行 print("{!<8}".format(1640),"年, 明亡!")#输出: 1640!!!! 年, 明亡!
第18行
第19行 print("{:.2f}".format(3.1415926))#输出: 3.14
第20行 print("{:05.2f}".format(3.1415926))#输出: 03.14
第21行 print('{0:e}'.format(3.1415926))#输出: 3.141593e+00
第22行
第23行 print('{:,}'.format(123456789)) #输出: 123,456,789
第24行
第25行 print("{0:b},{0:d},{0:o},{0:x}".format(19))#输出: 10011,19,23,13
第26行 #eof

```

#### (4) del函数

del函数更合适的说法是del 语句，因为del不是括号的形式出现，其功能用于删除变量或对象，如例程1-19所示。例程第6行删除变量a，如果此时使用a，则将报错，例程第7行所示。del还可以删除序列对象的组成部分，如例程第11、12行所示。一旦删除序列数据成员，序列将自动调整序列成员编号。如删除编号为0的成员，原编号为1的成员自动变成编号为0的成员，以此类推。

例程1-19

```

第1行 #del的使用示例
第2行 a=11
第3行 b="China"
第4行 listA=[12,13,14,234,3345,123]
第5行
第6行 del a
第7行 print(a)#本语句错误，一旦被删除，不能再被使用
第8行 #输出错误信息: name 'a' is not defined, 含义: 名称a没有被定义
第9行
第10行 del b
第11行 del listA[0]#删除listA中第一个数12，后续数据自动前移
第12行 del listA[2:5] #删除2,3,4，即删除最后三个数
第13行 print(listA)#仅仅输出[13,14]
第14行
第15行 del listA#删除listA，不能再被使用
第16行
第17行 #print(listA) #本语句报错，一旦被删除，不能再被使用
第18行
第19行 #eof

```

#### (5) help函数



help()函数用于查看函数或功能库使用说明，如图1-6所示，该函数主要在Python交互式环境下使用，图中help(print)即为查看print()函数的使用。熟练使用help()函数，对学习Python很有帮助。

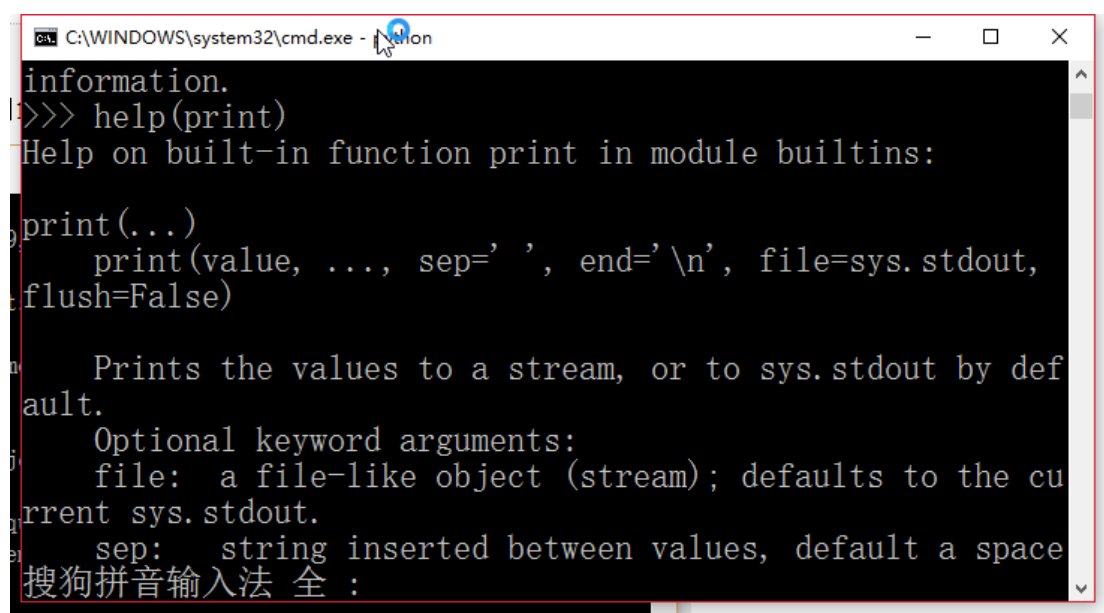


图1-9 help()函数的使用

**5、运算符**

Python大部分运算符与C/C++运算符很相近(不过没有条件运算符?:)。相较C/C++，Python有幂运算符\*\*和整除运算符//，如例程1-20所示。

例程1-20

第1行	#!/usr/bin/python
第2行	
第3行	a=5
第4行	b=3
第5行	
第6行	print(a/3)#output:1.66666666667
第7行	print(a//b)#output:1
第8行	
第9行	print(a**b)#output:125
第10行	
第11行	#eof
第12行	
第13行	

Python逻辑运算符是and(与)、or(或)和not(非)，而不是C/C++的&&(与)、||(或)和!(非)，如例程1-21。

例程1-21

第1行	#!/usr/bin/python
第2行	
第3行	a=5
第4行	b=3
第5行	print(a and b)#output:3
第6行	print(a or b)#output:5
第7行	print(not a)#output:False
第8行	
第9行	

```

第10行  a=5
第11行  b=False
第12行  print(a and b)#output:False
第13行  print(a or b)#output:5
第14行  print(not b)#output:True
第15行
第16行  #eof

```

Python还提供了标志算符is和is not，标志运算符做内存比较，如果内存位置相同则为True，否则为False。如例程1-22所示。

#### 例程1-22

```

第1行  #!/usr/bin/python
第2行
第3行  a=10
第4行  b=10
第5行  c=11
第6行  d=False
第7行
第8行  print(a is b)#output:True
第9行  print(a is c)#output:False
第10行 print(a is d)#output:False
第11行
第12行 def rtnTen(N):
第13行     return N-1
第14行
第15行 print(a is rtnTen(c))#output:True
第16行
第17行 #eof

```

in和not in是Python的成员运算符，如例程1-22所示。

#### 例程1-23

```

第1行  #!/usr/bin/python
第2行
第3行  a=[12,334,44,54]#a is list
第4行  b=44
第5行  print(b in a)#True
第6行  print(45 not in a)#True
第7行
第8行  #eof

```

### 6、功能扩展库

Python功能库非常丰富，可以说已经成为Python的重要特色，也是Python得以快速流行的主要原因。功能库又称扩展模块、扩展包等，其目的是扩展某个方面的功能，如数学模块(math)、随机模块(random)、网络模块(很多)等等。Python功能库可以分为标准库和第三方库，标准库在安装Python随之安装，第三方库则需要单独安装。安装成功后与标准库用法相同。在代码中使用功能库，需要使用关键字import，如例程1-24第4行所示。

#### 例程1-24

```

第1行

```

## #扩展模块的使用

```
第2行
第3行 #装入math扩展模块
第4行 import math
第5行
第6行 print(math.pi)#输出: 3.141592653589793
第7行
第8行 valA=math.floor(-11.98)#math.floor()返回小于等于参数的最大整数
第9行 print(valA)#输出: -12
第10行
第11行 valB=math.ceil(-11.98)#math.ceil()返回大于等于参数的最小整数
第12行 print(valB)#输出: -11
第13行
第14行 valB=math.factorial(5)#求5的阶乘
第15行 print(valB)#输出: 120
第16行
第17行 #math.radians()是将角度转换为弧长, 参数为角度
第18行 #math.cos()是求余弦, 其参数会为弧长
第19行 print(math.cos(math.radians(60)))#0.5000000000000001
第20行
第21行 print(math.sqrt(2))#求根号2的值, 此处输出: 1.4142135623730951
第22行
第23行 #eof
```

最常用的功能库还有random(随机库)和time(时间库), 例程1-25是random库的应用示例, 例程1-26是time库的应用示例。

### 例程1-25

```
第1行 #扩展模块的使用
第2行
第3行 #装入random扩展模块
第4行 import random
第5行
第6行 valA=random.random()#random.random()的功能是产生0-1之间的随机数
第7行 print("random.random()的使用: ",valA)#输出: 每次输出不一样
第8行
第9行 print("random.uniform(10,20)的使用: ",random.uniform(10,20))#产生10-20见的随机小数
第10行 print("random.randint(10,20)的使用: ",random.randint(10,20))#产生10-20见的随机整数
第11行
第12行 #从10开始, 每次增加2, 一直到50为止, 然后从中随机选取一个
第13行 print("random.randrange(10,50,2)的使用: ",random.randrange(10,50,2))
第14行
第15行 print("random.choice([1,3,4,1,7,9,11])的使用: ",random.choice([1,3,4,1,7,9,11]))#随机选取其中一个
第16行
第17行 listA=[1,3,4,1,7,9,11]
第18行 print("listA原始序列",listA)
第19行 random.shuffle(listA)#乱序
```

```

第20行 print("random.shuffle()后listA的序列: ",listA)#输出乱序后结果
第21行
第22行 print("random.sample()的使用: ",random.sample([1,3,4,1,7,9,11],3))#随机抽样3个
第23行
第24行 #eof

```

例程1-26使用扩展模块的方式与前述方式有所不同。from time import time的含义是“从time扩展模块中装入time”。在time模块中，除了time外，还有如sleep等函数。如果仅使用time()函数，则可from time import time，表明仅需装入time函数。如果写成import time，则time()的调用必须写为time.time()，sleep()函数写为time.sleep()。

例程1-26

```

第1行 #扩展模块的使用
第2行 from time import time
第3行
第4行 startTime=time()#获得起点时间
第5行
第6行 i=0
第7行 listA=[]
第8行 while i<1000000:
第9行     listA.append(i)
第10行     i+=1
第11行
第12行 endTime=time()#获得结束时间
第13行
第14行 print("循环总计耗时: ",endTime-startTime)#计算出循环消耗的时间
第15行 #在某笔记本上耗时输出: 0.2700822353363037，不同配置可能不同
第16行
第17行 #eof

```

除了标准库外，Python还有大量第三方库，需要安装才能使用。安装时运行“pip install 库名称”或“pip3 install 库名称”。当Python2.X和Python3.X同时存在时，运行pip3则安装Python3.X需要的库。如果仅有Python3.X，则pip或pip3功能相同。requests是常用第三方库，用于Web数据采集也就是从网站下载网页、各种文件等等，其安装命令是**pip install requests**或**pip3 install requests**，其中requests是库名称。当安装成功后，可进入Python交互环境，输入import requests如果能正常执行没有提示错误信息，则表示requests安装成功。例程1-27是requests应用示例。

例程1-27

```

第1行 import requests
第2行
第3行 r = requests.get('http://www.pku.edu.cn') #取得http://www.pku.edu.cn网址的内容
第4行 r.encoding="utf-8" #设置网页编码，此处一般为utf-8
第5行
第6行 print(r.text) #输出网页文本，r.text结果为字符串
第7行
第8行 #eof

```

## 第二节 控制语句

Python的流程控制语句非常简洁，没有很多语言中常见的for循环以及switch-case选择语句(如C/C++/C#、JavaScript、Java)。Python流程控制语句虽然非常简单，但也够用好用，甚至更胜一筹。

### 1、if-else选择语句

某些语句是否执行取决于是否满足指定条件，这种语句称之为选择语句，又称分支语句或者条件语句，Python的选择语句如例程1-28所示。第4行代码中的numA%2==0的含义是求得numA除以2的余数是否等于0，如果等于0，则执行第5行，否则执行第7行。第5行或第7行是有条件被执行，即由numA%2==0的结果确定。另外，else部分并不是必须，可根据需要决定取舍。

#### 例程1-28

```
第1行 #控制语句：if语句
第2行
第3行 numA=int(input("请输入一个整数："))
第4行 if numA%2==0: #注意：冒号不能省略
第5行     print(numA,"是一个偶数！")
第6行 else: #注意：冒号不能省略
第7行     print(numA,"是一个奇数！")
第8行
第9行 #eof
```

if除了有else子句外，还可以有elif子句，如例程1-29所示。例程1-30是例程1-29另外一种写法。在例程1-29中，elif和else是if的子句，地位相同，而在例程1-30中第9-12行由else子句控制，是选择语句的嵌套。

#### 例程1-29

```
第1行 #! /usr/bin/python
第2行
第3行 strNum =input("请输入一个整数")
第4行 intNum=int(strNum)
第5行
第6行 if num > 0:
第7行     print('这个数比0大')
第8行 elif num < 0:
第9行     print('这个数比0小')
第10行 else:
第11行     print('这个数是0')
第12行
第13行 #eof
```

#### 例程1-30

```
第1行 #! /usr/bin/python
第2行
第3行 strNum =input("请输入一个整数")
第4行 intNum=int(strNum)
第5行
第6行 if num > 0:
第7行     print('这个数比0大')
第8行 else:
第9行     if num<0:
第10行         print('这个数比0小')
第11行     else:
第12行         print('这个数是0')
第13行
第14行 #eof
```

一个if可以有多个elif部分，但只能有一个else，如例程1-31所示。

例程1-31

```
第1行  #!/usr/bin/python
第2行
第3行  nowSalary=500
第4行  titleRank=input("请输入职称: ")
第5行
第6行  if titleRank=="教授":
第7行      upValue=200
第8行  elif titleRank=="副教授":
第9行      upValue=150
第10行 elif titleRank=="讲师":
第11行     upValue=100
第12行 else:
第13行     upValue=50
第14行
第15行 print("你的薪水=",nowSalary+upValue);
第16行
第17行 #eof
```

## 2、for-in循环

当条件满足时某些语句被重复执行，控制其重复执行的语句称之为循环语句。在Python中，循环语句有for-in和while两种，例程1-32是for-in循环示例。和if-else相同，要注意冒号的使用，如第6行所示。

例程1-32

```
第1行  #控制语句：for-in
第2行
第3行  numSum=0
第4行
第5行  listA=[1,2,3,4,5,6]
第6行  for i in listA:#注意冒号，不能省略
第7行      numSum+=i
第8行
第9行  print(numSum)
第10行
第11行 #注：Python仅有这一种形式的for循环
第12行 #eof
```

连续有规律的整数序列，采用range()会更加方便，如例程1-33所示。

例程1-33

```
第1行  #for-in与range配合使用
第2行
第3行  numSum=0
第4行
第5行  for i in range(1,7):#注意冒号，不能省略
第6行      numSum+=i
第7行
```



```
print(numSum)#输出21
```

第8行

```
numSum=0
```

```
for i in range(1,101,2):#1,100之间的偶数
```

```
numSum+=i
```

```
print(numSum)#输出2500
```

第13行

```
#eof
```

### 3、while循环

除了for-in循环外，Python还提供了while循环，如例程1-34所示。

例程1-34

```
#控制语句： while
```

第2行

```
i=1
```

```
numSum=0
```

```
while i<=100:
```

```
numSum+=i
```

```
i+=1
```

第8行

```
print("1-100之间的和：",numSum)
```

第10行

```
#eof
```

在循环语句for-in和while中，还常使用break和continue。break用于终止break所在当前循环，continue用于停止本次循环直接进入下一次循环。如例程1-35所示。在第5-9行中，第6-7行的含义是当i的值为5时break，即终止循环，5以后的不会被处理，因此第9行仅输出到4。在第13-17行中，第14-15行的含义是当i值为5时，当次循环停止直接进入下一次循环，即5时不处理，进入下一次6的循环，因此第18行仅仅没有输出5。

例程1-35

```
#!/usr/bin/python
```

第2行

```
#循环中break和continue的使用
```

第4行

```
for i in range(1,10):
```

```
if i==5:
```

```
break
```

第8行

```
print(i,end=" ")
```

```
#上述循环执行后输出： 1 2 3 4
```

第11行

第12行

```
for i in range(1,10):
```

```
if i==5:
```

```
continue
```

第16行

```
print(i,end=" ")
```

第18行 #上述循环执行后输出：1 2 3 4 6 7 8 9，缺5

第19行

第20行 #eof

循环语句还可以和else语句联合使用，如例程1-36，从中可以看出当循环正常结束时，将会执行else子句所控制代码，如果中途被break语句终止，则不会执行else子句所控制的代码。for-in循环同样支持else子句，如例程1-37所示。

#### 例程1-36

第1行 #控制语句：if语句

第2行 i=3

第3行 while i<99:

第4行 j=2

第5行 while j<i:

第6行 if i % j==0:

第7行 break

第8行 else:

第9行 j+=1

第10行 else:

第11行 print(i,"是一个质数! ")

第12行

第13行 i+=1

第14行 else:

第15行 print("循环结束! ")

第16行

第17行 #eof

#### 例程1-37

第1行 #for-in循环的else子句

第2行 for num in range(10,20): #介于10到20之间的数字

第3行 for i in range(2,num): #介于2到num之间

第4行 if num%i == 0:#确定第一个质因数

第5行 j=num/i# 计算第二个质因数

第6行 print('%d 等于 %d \* %d' % (num,i,j))

第7行 break # 跳出当前循环

第8行 else:# 循环的 else 部分

第9行 print(num, '是一个质数')

第10行

第11行 print("外层for-in循环结束! ")

第12行 #eof

### 第三节 Python的缩进

在Python中，被控制语句必须缩进，可以采用空格或Tab缩进，且同位置同等级受控语句必须采用相同方式缩进。缩进时，空格和Tab不能混用例程1-38是有关缩进的更多示例。

#### 例程1-38

第1行 #! /usr/bin/python

第2行 #Python缩进示例

第3行

```

第4行 import math#装载入math功能库
第5行
第6行 a,b,c=1,3,1#a,b,c为一元二次方程的系数
第7行 delta=b*b-4*a*c
第8行 if delta<0:
第9行     print("没有实数根")
第10行 else:
第11行     if delta==0:
第12行         print("有两个相等实数根")#3个Tab缩进
第13行         print("x1=x2=", -b/(2*a))#3个Tab缩进
第14行     else:
第15行         print("有两个不相等实数根")#采用空格缩进
第16行         print("x1=", (-b+math.sqrt(delta))/(2*a), "x2=", (-b-math.sqrt(delta))/(2*a))
第17行     #eof

```

例程1-38是选择语句嵌套，采用了多种缩进形式。第12、13行采用多个Tab缩进，而第15、16行采用空格缩进。第12、13行与第15、16行等级相同，都是选择用于控制的下一层，但在不同位置。第12、13行受if控制而第15、16行受else控制，因此两者之间的缩进并不相同。但第12、13行两行，则位置相同等级相同，则必须采用相同的缩进，第15、16行与之同理。

## 第四节 函数定义

Python提供了一些内置函数，如print()、input()、int()、float()等等，但内置函数毕竟有限，只能提供基本通用函数，难以满足更多个性化需要，因此自定义函数就成为几乎所有程序设计语言的必须提供功能。Python也提供自定义函数功能，如例程1-39第3-7行所示。**自定义函数以def关键字开始**，如第3行所示。

例程1-39

```

第1行 #Python的自定义函数
第2行
第3行 def myAbs(x):
第4行     if x<0:
第5行         return -x
第6行     else:
第7行         return x
第8行 numX=-10
第9行 numA=myAbs(numX)
第10行 print("|",numX,"|=",numA)
第11行
第12行 numB=myAbs(numA)
第13行 print("|",numA,"|=",numB)
第14行
第15行 #eof

```

在def myAbs(x)中，myAbs成为函数的名称，x称为函数的参数。函数参数可以有多个用逗号分开，也可以有一个，甚至可以一个都没有。即便没有参数，圆括号也不能省略。第9行numA=myAbs(numX)中的myAbs(numX)称之为函数调用。函数调用必须有括号，即便没有参数。调用时参数个数与函数定义相关。定义函数时的参数称之为形式参数(形参)，调用函数时的参数称之为实际参数(实参)。形参和实参无关，定义是x调用numX。

对于例程1-39如果没有定义myAbs()函数，则第9、12行要使用该功能都必须重复撰写相同代码。不但增加代码量，也增加维护难度。将一个复杂任务，分解为多个可以实现的函数，是程序设计的重要训练。一旦分解为多个功能不同的函数，就能实现一定程度的分工合作，有利于发挥协同力量。

一般说来，函数返回通常为一个，但Python函数还支持返回多个值，在返回时自动将返回值封装为Tuple，如例程1-40所示。从第8-9行能观察出，Python的多返回值实际上是Tuple类型。

#### 例程1-40

```
第1行  #! /usr/bin/python
第2行
第3行  #DivMod()返回两个参数，返回值类型为tuple
第4行  def DivMod(n,m):
第5行      return n//m,n%m #返回两个整数相除的商和余数
第6行
第7行  result=DivMod(11,5)
第8行  print(result) #输出:(2,1)
第9行  print(type(result)) #输出: <class 'tuple'>
第10行
第11行  #eof
```

例程1-41也是一个函数定义，其中pass是占位符，不起实质作用。另外，当一个函数没有用return返回值时，系统默认其返回值为None。pass占位符不仅可以函数，也可以用于流程控制语句。

#### 例程1-41

```
第1行  #! /usr/bin/python
第2行
第3行  def Foo():
第4行      pass
第5行
第6行  print(Foo())#print:None
第7行
第8行  #eof
```

函数定义时可以指定参数默认值，如例程1-42所示。ADD()函数本有两个参数，但第4行调用时只有一个参数，另外一个参数用其默认值。当有两个参数时，则应用给定值。

#### 例程1-42

```
第1行  def ADD(N,plusNum=1):
第2行      return N+plusNum
第3行
第4行  a=ADD(10)
第5行  print(a) #输出11， plusNum此时默认值为1
第6行
第7行  b=ADD(10,20)
第8行  print(b) #输出30， plusNum此时为20
第9行
第10行  #eof
```

例程1-43第4行SUM()函数的参数和前面的函数定义不同，参数前面多了英文星号，此时该变量名称可以代表多个值，是为变参，可以用for-in循环遍历其每个值，如例程第6行所示。第10行、第13行分别SUM()函数处理3个参数和6个参数。

例程1-43第17行的函数student()定义又有所不同，Math参数前有英文星号，但星号与Math之间有空格，而变参函数的参数前没有空格。student()函数的Math、Chinese以及Foreign参数称之为命名参数。命名参数的调用顺序与定义可以不一致，如例程第20行所示。在Python中，print()函数的end和sep即为命名参数。

#### 例程1-43

```

第1行  #变参函数定义,命名参数函数定义
第2行
第3行  #变参函数定义
第4行  def SUM(*num):
第5行      result=0
第6行      for i in num:
第7行          result+=i
第8行      return result
第9行
第10行  sumResult=SUM(11,23,7) #3个参数
第11行  print(sumResult) #输出:41
第12行
第13行  sumResult=SUM(-11,23,7,22,100,99) #6个参数
第14行  print(sumResult) #输出:240
第15行
第16行  #命名参数函数定义
第17行  def student(studName,* ,Math,Chinese,Foreign): #studName为普通参数
第18行      print("{}的成绩,数学={},语文={},外语={}".format(studName,Math,Chinese,Foreign))
第19行
第20行  student("李白",Chinese=100,Math=80,Foreign=100)
第21行
第22行  #eof

```

有关Python函数的更多细节，将在后续相关章节讲解。

## 第五节 lambda函数

lambda函数一种匿名函数，目前很多高级语言都有lambda函数，例程1-44是Python中Lambda函数的使用。第2-3行是常规函数定义，第6行是Lambda函数定义，两相比较，lambda关键字之后冒号之前的x相当于参数，x+2相当于return x+2。第7-8行是常规函数和lambda函数的使用。仅从第7-8行，难以看出lambda函数的优势，其常用于序列操作函数如：map、filter等等，如第16行所示。代码map(lambda a:a\*2,listA)的功能是将listA中的每个成员的值乘以2。该功能很简单，为此单独写一个函数且也仅在此处使用，很是没有必要。即便定义函数能实现相同功能，但根据函数名称药知其定义，需离开改行，远不如lambda函数直观了然。

例程1-44

```

第1行  #Lambda的定义及其使用
第2行  def Add2(x):#通常函数定义
第3行      return x+2
第4行
第5行  #Lambda函数定义
第6行  myFunc=lambda x:x+2
第7行  valExecA=myFunc(10)
第8行  valExecB=Add2(10)
第9行
第10行  print(valExecA)#输出： 12
第11行  print(valExecB)#输出： 12
第12行
第13行  #lambda函数的应用

```

第14行	listA=[1,3,5,2,12]
第15行	print(listA)
第16行	listB=list(map(lambda a:a*2,listA))
第17行	print(listB)
第18行	
第19行	#eof

另外，Python的lambda函数分号后不能有语句或注释，只能是表达式，该表达式即为lambda函数根据参数计算出的返回值。

第六节 序列类型

序列数据类型是Python的重要特征，也可以说是Python的特色。在Python中，常用序列数据类型有：list(列表序列)、Str(字符串序列)、Range(规则整数序列)、Dict(字典序列)、Tuple(元组序列)、Set(集合序列)等。序列类型可以分为可变序列和不可变序列，可变序列可以修改序列中已有数据，而不可变序列则不可修改。list、Dict和Set是可变序列，而Str、Range、Tuple是不可变序列。

1、Str(字符序列)

字符串常用于与文本有关的处理，几乎所有高级语言都有字符串处理能力，但和其他不少高级语言相比，Python字符串处理能力相当强大。在Python中，字符串用配对的单引号、双引号、三引号界定。在例程1-45中，第7行以双引号界定，第9行以单引号界定。在单引号或双引号界定的字符串中，还可以包含引号，不过需要注意配对关系。单引号字符串中，可以一个或者多个双引号，反之亦成立。

在第9行、第12行代码中，还使用了转义字符，即\n表示的不是两个字符的组合，而是另有含义，\n代表换行。

在Python中，除了单引号和双引号外，还有比较少见的三引号，用于表示多行文本。三引号可以是三个单引号对，也可以是三个双引号对。例程第15-19行代码是三个单引号，换成三个双引号亦可。

例程1-45

第1行	#str应用示例
第2行	
第3行	print(str(5.05))#将数值转换为字符串
第4行	
第5行	strA=""#空字符串
第6行	
第7行	strB="Great China!"#双引号字符串
第8行	
第9行	strC='Python\'s philosophy rejects the Perl "there is more than one way to do it" approach to ...'#单引号字符串
第10行	print(strC)
第11行	
第12行	strD="Beautiful is better than ugly\n Explicit is better than implicit \nSimple is better than complex"#转义字符的使用
第13行	print(strD)
第14行	
第15行	strE="""Beautiful is better than ugly
第16行	Explicit is better than implicit
第17行	Simple is better than complex
第18行	Complex is better than complicated
第19行	Readability counts"""#三单引号字符串
第20行	print(strE)
第21行	



与前述list类似，在字符串操作中，也经常需要获取一个片段，或者称为子串。Python的子串操作尤为方便，如例程1-46所示。第8行代码中strA[0]的含义是取得字符串变量strA中编号为0的元素。和其他很多语言相同，Python字符串的每个元素的编号也是从0开始，因此strA[0]实际上取得strA中第一个元素。同样是第8行，strA[-2]的含义是取出倒数第2个元素，最后一个元素的编号为-1。

例程1-46

```

第1行  #! /usr/bin/python
第2行
第3行  #str应用示例
第4行
第5行  strA="图灵，人工智能之父。"
第6行  print(strA)
第7行
第8行  print(strA[0],strA[-2],sep="$$$")#输出：图$$$父
第9行  print(strA[3:5],strA[0:2],strA[:7],strA[3:],sep="$$$")
第10行 #上行输出：人工$$$图灵$$$图灵，人工智能$$$人工智能之父。
第11行
第12行 print(strA[-7:-1],strA[3:-3],sep="$$$")#输出：人工智能之父$$$人工智能
第13行
第14行 strB="ABCDEFGFG"
第15行 print(strB[0:5],strB[1:5:2],strB[-1::-2],sep="$$$")#输出：ABCDE$$$BD$$$GECA
第16行 print(strB[5:1:-1])#输出：FEDC
第17行
第18行 print("ABC" in "ABCDEFGFG")#包含判断，输出：True
第19行
第20行 #eof

```

在Python中，切片(slice)是很重要的操作，如例程第9行所示，strA[3:5]表示获取从编号为3的元素开始到编号为5的元素为止（不包含编号为5的元素）。strA[0:2]与之类似。在strA[:7]和strA[3:]中，分别省略了起点和终点。当省略起点时，默认从编号0开始；省略终点时，默认从指定编号开始直到最后。例程第12行是切片的另外一些示例。

切片的语法格式为：**start\_index:end\_index:step**，即切片参数可以有三个。strB[1:5:2]的作用是从编号1开始，隔1个字符取1个。step的含义间隔截取，间隔数量为step-1。切片参数可以省略。当省略start\_index时，默认从编号0开始；当省略end\_index时，默认到最后一个元素；当省略step默认为1。注意：step不能为0。

字符串下标操作可认为是一种特殊的切片操作，其目的是获得字符串的一个局部。在字符串操作中，还经常判断字符串是否被包含在另外一个字符串之中，如第18行所示，用in即可。反之可用not in判断。

对于字符串对象，+用于联接两个字符串，\*用于重复，如例程1-47所示。值得注意的是：执行第14行代码strA[0]="X"将导致错误，其原因是str是不可变序列，而list是可变序列。此处的不可修改是指strA所代表的字符串不可被修改，此处所代表对象为"ABCD"，不能被修改为XBCD。虽然"ABCD"不可被修改，但是strA可以被修改为其他，如第7行、第10行所示。当strA不在代表"ABCD"时，该字符串所占据内存成为垃圾内存，会自动从内存中消失。

例程1-47

```

第1行  #! /usr/bin/python
第2行
第3行  #str应用示例
第4行  strA="ABCD"
第5行  strB="01234567"
第6行  print(strA+strB)#字符串联接，此处输出：ABCD01234567

```

```

第7行    strA=strA+strB
第8行    print(strA)#输出: ABCD01234567
第9行
第10行   strA="ABCD"*3#重复3遍
第11行   print(strA)#输出: ABCDABCDABCD
第12行
第13行   #下行代码删除#将导致错误, 字符串不可被修改
第14行   #strA[0]="X"
第15行
第16行   for c in strA:
第17行       print(c)
第18行
第19行   #eof

```

str类型还有其他一些常见操作, 如例程1-48所示, 更多操作列入表1-1中。len()方法返回字符串的长度, 如第5行代码所示; split()方法用于按照设定字符切分, 结果为list类型; find()用于查找子串出现位置, 如第10行所示; replace()用于字符串替换, 如第13行所示(注意: 原字符串组成没有发生变化); strip()用于删除字符串两端空格; lstrip()用于删除左侧空格;rstrip()用于删除右侧空格; join()用于字符串穿插, 如第22行所示(注意: strA.join(strB)是将strA穿插在strB的每个字符之间); ord()用于返回每个字符的序号; chr()将序号转化为字符。

#### 例程1-48

```

第1行    #! /usr/bin/python
第2行    #str应用示例
第3行
第4行    strA="ABCD,0123,循序渐进,O K "
第5行    print(len(strA))#输出: 17
第6行
第7行    listA=strA.split(",")
第8行    print(listA)#输出: ['ABCD', '0123', '循序渐进', 'O K']
第9行
第10行   xxPos=strA.find("循序")
第11行   print(xxPos)#输出: 10
第12行
第13行   print(strA.replace("12","壹贰"),strA,sep="$$")
第14行   #上行输出: ABCD,0壹贰3,循序渐进,O K $$ABCD,0123,循序渐进,O K
第15行
第16行   strB=" Y E S "#YES前后各有两个空格, 每个字符间有空格
第17行   print("XXX",strB.replace(" ",""),strB.strip(),strB.rstrip(),strB.lstrip(),"ZZZ",sep="$$")
第18行   #上行输出: XXX$$YES$$Y E S$$ Y E S$$Y E S $$$ZZZ
第19行
第20行   strC="0123XYZ"
第21行   strD="壹贰叁肆"
第22行   print(strC.join(strD))#输出: 壹0123XYZ贰0123XYZ叁0123XYZ肆
第23行
第24行   print(ord("A"),ord("中"),sep="$$")#输出: 65$$20013
第25行   print(chr(65),chr(20013),sep="$$")#输出: A$$中
第26行

```

表1-1: Str对象常用方法及其相关函数

方法名称及其功能	功能描述	示例
strO.center(widths[,fillchar]) 两侧填入字符	以strO为中心，两侧填入fillChar设定的单个字符直到长度为width值。如省略fillChar参数，则默认为空格	print("ABCD".center(7,"XX")) #输出: 'XXABCDX'
strO.count(subStr[,start[,end]]) 统计subStr在strO出现次数	统计从start开始到end截止，subStr在strO中出现的次数。	print("ABCDABCDab".count("AB")) 输出: 2 print("ABCDABCDab".count("AB",1)) 输出: 1
strO.find(subStr[,start[,end]]) 在strO中查找子串substr	从start开始截止end，在strO中查找子串substr。省略start则从头查找，省略end则查找找到结束为止。找到第1个匹配后终止查找。返回值为第一次出现位置，如果未找到返回值为-1。	print("ABCDABCD".find("BC")) 输出: 1 print("ABCDABCD".find("BC",2)) 输出: 1
strO.rfind(subStr[,start[,end]]) 在strO中反向查找	与strO.find()类似，不过查找最后一次出现为止。	print("ABCDABCD".rfind("BC")) 输出: 5
strO.index(subStr[,start[,end]])	与find()类似，但当找不到subStr时，触发ValueError异常。	print("ABCDABCD".index("BC")) 输出: 1
strO.rindex(subStr[,start[,end]])	与rfind()类似，但当找不到subStr时，触发ValueError异常。	print("ABCDABCD".rindex("BC")) 输出: 5
strO.startswith(suffix[, start[, end]])	判断strO所代表的字符串是否以suffix开始，如是则返回True，否则False。可以设定start以及end，如设定则以start开始或以end结束。	print("zzAzABCAzzA".startswith("zzA")) 输出: True
strO.endswith(suffix[, start[, end]])	判断strO所代表的字符串是否以suffix结尾，如是则返回True，否则False。可以设定start以及end，如设定则以start开始或以end结束。	print("zzAzABCAzzA".endswith("zA")) 输出: True
strO.expandtabs(tabsize=8)	设定strO所代表的字符串\t所占据字符数，默认为8。	print("0123456789".expandtabs(2)) 输出: 0 123 456789
strO.join(iterable)	用strO将iterable所代表的成员连接成一个字符串。如果iterable没有字符串值则将产生TypeError异常。	print("_".join(["唐诗","宋词","元曲"])) 输出: 唐诗_宋词_元曲
strO.translate(transTable)	用strO按transTable设定映射关系进行转换。transTable由str.maketrans()产生。	transTab=str.maketrans("0123456789","零壹贰叁肆伍陆柒捌玖") print("879".translate(transTab)) #输出: 捌柒玖
str.maketrans(x[,y])	maketrans()用于生成str.translate(transTable)中的transTable即转换对照表。当仅有一个参数时，x必须为dict；当有两个参数时，必须为两个等长的字符串，用于指定映射关系。	transTab=str.maketrans("0123456789","零壹贰叁肆伍陆柒捌玖") print("879".translate(transTab)) #输出: 捌柒玖
strO.split(separator,maxsplit=-1)	以separator切分strO代表的字符串，当separator为空字符串时，则将strO代表的字符串逐一切分。返回值类型为list。当省略maxsplit或maxsplit=-1时，则全部内容都被切分。如maxsplit为具体值，则将strO切分为指定个数，未被切分部分为整体。	print("AzzBzzzCzzD".split("zz")) #输出: ['A', 'B', 'zC', 'D'] print("AzzBzzzCzzD".split("zz",2)) #输出: ['A', 'B', 'zCzzD']
strO.rsplit(separator,maxsplit=-1)	与strO.split()相似，不过切分时从右到左，	print("AzzBzzzCzzD".split("zz"))

	而strO.split()则是从左到右。	#输出: ['A', 'Bz', 'C', 'D']
strO.splitlines([keepends])	行切分, 形成行的list。将字符串从分行位置处切分形成list。分行位置包括但不限于: \n、\r\n、\v、\f等。	print("蒹葭苍苍, \n白露为霜。 \n所谓伊人, \n在水一方。".splitlines()) #输出: ['蒹葭苍苍, ', '白露为霜。', '所谓伊人, ', '在水一方。']
strO.replace(old, new[, count])	将strO中指定字符old替换为指定字符new。如果指定count则从左到右指定个数count被替换。	print("AzzBzzzCzzD".replace("zz", "xyz")) #输出: AxyzBxyzzCxyzD print("AzzBzzzCzzD".replace("zz", "xyz", 2)) #输出: AxyzBxyzzCzzD
strO.partition(separator)	用separator切分strO所代表的字符串, 分隔处为strO中第一次出现separator的位置。返回值为3元素tuple, 第一个为separator左边的内容, 第二个为separator第三个为separator右边的内容。如果strO中没有separator则第一个元素为strO自身, 第二和三个元素均为空白。	print("大江东去, 浪淘尽, 千古风流人物".partition(", ")) 输出: ('大江东去', ', ', '浪淘尽, 千古风流人物')
strO.rpartition(separator)	用separator切分strO所代表的字符串, 分隔处为strO中最后一次出现separator的位置。返回值为3元素tuple, 第一个为separator左边的内容, 第二个为separator, 第三个为separator右边的内容。如果strO中没有separator则第一和二元素为空白, 第三个元素为strO自身。	print("大江东去, 浪淘尽, 千古风流人物".rpartition(", ")) 输出: ('大江东去, 浪淘尽', ', ', '千古风流人物')
strO.upper()	将strO代表的字符串全部转换为大写。	print("China is Great!".upper()) 输出: CHINA IS GREAT!
strO.lower()	将strO代表的字符串全部转换为小写。	print("China is Great!".upper()) 输出: china is great!
strO.zfill(nWidth)	构造长度为n的字符串, 在原字符串strO之前填充多个0以达到长度n。对非数字字符串同样适用。	print("123".zfill(5)) #输出: 00123 print("-123".zfill(5)) #输出: -0123
strO.ljust(width[, fillchar])	构造长度为width的字符串, 如果strO的长度小于width, 则strO尾部用fillchar填充以达到指定长度。如strO长度大于width, 则不调整。当省略fillchar则默认为空格。	print("123".zfill(5)) #输出: 00123 print("-123".zfill(5)) #输出: -0123
str.rjust(width[, fillchar])	构造长度为width的字符串, 如果strO的长度小于width, 则strO头部用fillchar填充以达到指定长度。如strO长度大于width, 则不调整。当省略fillchar则默认为空格。	print("123".zfill(5)) #输出: 00123 print("-123".zfill(5)) #输出: -0123
strO.title()	将strO所代表的英文单词首字母转换为大写。不能对汉语或数字转换。	print("China is great!".title()) #输出: China Is Great!
strO.swapcase()	将strO所代表的英文单词中的大写英文字母转换小写, 小写字母转换为大写。不能对汉语或数字转换。	print("China is great!".swapcase()) #输出: cHINA IS GREAT!
strO.strip([chars])	将strO所代表的字符串两侧的空格(此时没有参数)或指定字符删除。	print(" ABC ".strip()) #输出: ABC print("zzAzABCAzzA".strip("zA")) #输出: BC, 相当于删除两侧的z或A。
strO.lstrip([chars])	将strO所代表的字符串左侧的空格(此时没有参数)或指定字符删除。	print(" □□ABC□□ ".lstrip()) #输出: ABC□□ print("zzAzABCAzzA".lstrip("zA")) #输出: BCAzzA, 相当于删除左侧的z或A。
strO.rstrip([chars])	将strO所代表的字符串右侧的空格(此时没有参数)或指定字符删除。	print(" □□ABC□□ ".rstrip()) #输出: □□ABC print("zzAzABCAzzA".rstrip("zA"))

		#输出: zzAzBC, 相当于删除右侧的z或A。
strO.capitalize()	将strO所代表的字符串首字母转换为大写。	print("china is great!".capitalize()) #输出: China is great!
strO.isnumeric()	如果strO所代表的字符串全部是数字, 则返回值为True。如有小数点或正负号则返回False。	print("123".isnumeric()) #输出: True
strO.isdigit()	如果strO所代表的字符串全部是数字, 则返回值为True。如有小数点或正负号则返回False。	print("123".isdigit()) #输出: True
strO.isalnum()	如果strO所代表的字符串只包含字母和数字, 则返回True, 否则返回False。	print("123ABC".isalnum()) #输出: True
strO.isalpha()	如果strO所代表的字符串只包含字母, 则返回True, 否则返回False。	print("abcxyz".isalpha()) #输出: True
strO.islower()	如果strO所代表的字符串全部是小写字母, 则返回True, 否则返回False。	print("abcxyz".islower()) #输出: True
strO.isupper()	如果strO所代表的字符串全部是大写字母, 则返回True, 否则返回False。	print("ABX".isupper()) #输出: True
strO.isdecimal()	如果strO所代表的字符串构成十进制数值, 则返回True, 否则返回False。	print("123".isdecimal()) #输出: True
strO.isspace()	如果strO所代表的字符串全部是空格, 则返回True, 否则返回False。	print(" ".isspace()) #输出: True
strO.istitle()	如果strO所代表的字符串中单词首字母大写, 则返回True, 否则返回False。	print("Hello,China!".istitle()) #输出: True
strO.isprintable()	如果strO所代表的字符串中所有字符均可打印, 则返回True, 否则返回False。不可打印字符由系统定义, 一般说来分隔符(\t、\n、\r等)均是不可打印, 但空格除外。	print("ABC\txyz".isprintable()) #输出: False

2、list(列表序列)

list是Python中最常用的类型, 前面已经有所涉及, 例程1-49是list类型的更多应用示例。Python的list类型与C/C++的数组类似, 但比C/C++数组更便于使用。list类型可以采用方括号访问数组成员, 如listA[0]表示访问编号为0的元素, 即第一个元素。和其他语言不同, 可以用负号方式反向访问成员, 如listA[-1]访问最后一个元素, listA[-2]访问倒数第二个元素。list是可变序列, 可以直接修改数据组员。append()追加新元素如第11-13行所示, insert()在指定位置插入新元素如第26行所示。pop()删除最后一个元素如第18行所示, 并可指定位置删除如第20行所示。

例程1-49

```
第1行  #! /usr/bin/python
第2行
第3行  #序列数据类型, list应用示例
第4行
第5行  listA=[1,'123','XYZ',12.12] #注意: list可以将不同类型的数据放在一起, 其他序列类型也如此
第6行  print(listA[0])#输出: 1
第7行  print(listA)#输出: [1, '123', 'XYZ', 12.12]
第8行
第9行  print(listA[-1])#输出最后一个元素, 此处输出12.12
第10行 print(listA[-2])#输出倒数第二个元素, 此处输出XYZ
第11行
第12行 listB=[] #空list
第13行 listB.append(100)
```

```

第14行 listB.append("ABC");
第15行 listB.append(11.11);
第16行
第17行 print(listB[1])#输出: ABC
第18行 print(listB)#输出: [100, 'ABC', 11.11]
第19行
第20行 listA.pop()#删除最后一个元素
第21行 print(listA)#输出: [1, '123', 'XYZ']
第22行 listA.pop(1)#删除编号为1的元素
第23行 print(listA)#输出: [1, 'XYZ']
第24行
第25行 listC=["Apple","Peach","Banana","Cherry","Ginkgo","Grape","lichee"]
第26行 listC.remove("Ginkgo") #按值移走, 如果有多个Ginkgo则仅移走满足要求的第1个
第27行 print(listC)#输出: ['Apple', 'Peach', 'Banana', 'Cherry', 'Grape', 'lichee']
第28行
第29行 del listC[1:4]#删除编号为1、2、3元素
第30行 print(listC)#输出: ['Apple', 'Grape', 'lichee']
第31行 del listC#删除listC变量, listC不存在
第32行 #print(listC) #本行执行错误, listC已经被删除
第33行
第34行 listB.clear()#输出: 清除listB中的全部元素
第35行 print(listB)#输出: [] listB为空的list
第36行
第37行 print(len(listA))#len求元素个数, 此处输出: 2
第38行
第39行 listA.insert(1,"AAA")#插入元素编号为1
第40行 print(listA)#输出: [1, 'AAA', 'XYZ']
第41行
第42行 #eof

```

排序用途广泛, 例程1-50是list的sort()应用示例。第3、5、6行是list默认按升序排序。在排序时, 可以指定是否反序, 如例程第8-9行所示。第11行中的nameScorelist中包含姓名和成绩, 如果按成绩排序, 则需要指定key(排序键或者说排序项目), 如例程第12行所示, 其功能是指定排序时所依赖的值。排序时可以指定排序键以及是否反序, 如第15行所示。

#### 例程1-50

```

第1行 #! /usr/bin/python
第2行
第3行 scorelist=[80,70,98,76]
第4行
第5行 scorelist.sort() #对list排序
第6行 print(scorelist) #[70, 76, 80, 98], 升序
第7行
第8行 scorelist.sort(reverse=True) #输出: 反序
第9行 print(scorelist) #输出: [98, 80, 76, 70]
第10行
第11行 nameScorelist=[["杨杰",80],["王帅",70],["李慧",98],["王雪",76]]
第12行

```



```

第13行 nameScorelist.sort(key=lambda x:x[1])
第14行 print(nameScorelist);#输出: [['王帅', 70], ['王雪', 76], ['杨杰', 80], ['李慧', 98]]
第15行
第16行 nameScorelist.sort(key=lambda x:x[1],reverse=True)
第17行 print(nameScorelist) #输出: [['李慧', 98], ['杨杰', 80], ['王雪', 76], ['王帅', 70]]
第18行 #eof

```

排序根据键值。当没有指定参数时，默认根据数据成员的值进行比较。如果成员值不能比较，则将触发异常。当通过key制定排序函数时，则排序按函数计算结果进行排序，但并不改变list成员值。执行例程1-51第5行将到值错误，因为排序必然进行数据成员的比较，整数不能直接与字符串进行比较，将导致错误。第8行将成员值转换为整数，此时可以进行比较。当指定key即指定排序函数时，函数将逐一处理每个数据成员，然后根据处理后的值进行排序，如例程第12行所示，对每个成员值求余数，对余数进行排序。注意：函数值仅用于排序，不改变成员值本身，从第14行的输出可知。

#### 例程1-51

```

第1行 #排序例程
第2行
第3行 listA=[12,23,"34","56","21",111]
第4行
第5行 #listA.sort() #删除行首的#将导致错误
第6行 #将导致排序失败，字符串与数值不能比较大小
第7行
第8行 listA.sort(key=lambda x:int(x)) #转换为整数
第9行 print(listA) #输出: [12, '21', 23, '34', '56', 111]
第10行
第11行 listB=[12,23,34,56,111,123,321]
第12行 listB.sort(key=lambda x:x%2) #排序后偶数在前，奇数在后
第13行 #当成员值为偶数时，x%2的值为0，否则为1
第14行 print(listB) #[12, 34, 56, 23, 111, 123, 321]
第15行
第16行 #eof

```

当对list执行remove()和count()时，将对list成员从到尾进行比较然后执行相应操作。在例程1-52第6、7行以及第19、20行将大量产生0-4之间的值，然后删除4，有两种不同的执行策略，时间差别极大。第9-10行的方式判断4是否存在于list，如果存在则删除。每次执行while将判断4是否在list之中，如碰倒第1个，则删除第1个4，然后继续执行。没删除一个元素，都将执行两次从头开始查找的过程，找到后删除(删除时还将执行一系列操作)。第22-23行则是找到不是4的成员则直接添加到listC之中，一次循环即可，而append()执行效率较高。在笔者电脑，第1种方法耗时22秒，第2种0.04秒，相差巨大。当然采用第二种方法，增加了listC，将浪费一些内存空间，相当于优化时间浪费空间。例程还提供第3中方法，执行时间是0.33秒，达到了一定程度时空平衡，可以参考。

#### 例程1-52

```

第1行 #执行效率比较
第2行 from time import time
第3行
第4行 startTime=time() #开始时间
第5行 listA=[]
第6行 for i in range(0,100000):
第7行     listA.append(i%5) #大量产生0,1,2,3,4
第8行

```

```

第9行 while 4 in listA:
第10行     listA.remove(4)
第11行
第12行 endTime=time()#结束时间
第13行 print(endTime-startTime,len(listA))
第14行
第15行 startTime=time() #开始时间
第16行 listB=[]
第17行 listC=[]
第18行
第19行 for i in range(0,100000):
第20行     listB.append(i%5) #大量产生0,1,2,3,4
第21行
第22行 for i in listB:
第23行     if i!=4:listC.append(i)
第24行
第25行 endTime=time() #结束时间
第26行 print(endTime-startTime,len(listC))
第27行
第28行 startTime=time() #开始时间
第29行 listD=[]
第30行 for i in range(0,100000):
第31行     listD.append(i%5) #大量产生0,1,2,3,4
第32行
第33行 memCount=len(listD)
第34行 for i in range(memCount-1,-1,-1):
第35行     if listD[i]!=4:del listD[i]
第36行
第37行 endTime=time()#结束时间
第38行 print(endTime-startTime,len(listD))
第39行 #eof

```

### 3、Tuple(元组序列)

在Python中，Tuple类型也比较常用，如例程1-53所示。和list相比，Tuple序列的数量不可变更，即不能对Tuple序列中元素的个数进行调整，不能增加新元素，也不能减少已有元素，更不能插入新元素。如果是相同数据组成，Tuple相对于list一般说来速度更快且占用内存空间更少。注意：第5行代码是将多个值赋给一个变量，其本质是将一个Tuple赋值给变量，不过强烈不建议如此赋值，建议第3行模式为好。

**例程1-53**

```

第1行 #Tuple(元组)的使用
第2行
第3行 tupleA=(12,23,"ABC","XYZ",12.12,23.11)
第4行
第5行 tupleB=12,23,"ABC","XYZ"
第6行 print(type(tupleB))#输出: <class 'tuple'>
第7行
第8行 tupleC=()#空的tuple

```

```

第9行 tupleD=(12,)#当申明仅有一个元素的Tuple时，必须加上逗号以规避歧义
第10行
第11行 print(tupleA[2])#输出：ABC
第12行
第13行 numX=tupleA[1:4]
第14行 print(numX)#输出：(23, 'ABC', 'XYZ')
第15行
第16行 tupleE=tupleA+tupleB#用+将另一个Tuple拼接在一起
第17行 print(tupleE)
第18行
第19行 del tupleE#删除tupleE，del后tupleE不能再被使用
第20行
第21行 tupleX=(1,4,1,24,43)
第22行 print(max(tupleX))#输出：43
第23行 print(min(tupleX))#输出：1
第24行 print(len(tupleX))#输出：5
第25行
第26行 listX=[1,2,3,4]
第27行 tuplelist=tuple(listX)#将list转换为Tuple
第28行
第29行 #eof

```

#### 4、Set(集合类型)

Set类型用于无重复值序列，可用一对花括弧申明，如例程1-54第3-4行所示。第8-9行则是通过add()和remove()方法对Set类型增加或者删除成员。对于Set类型，可以进行各种集合运算，如：交集、差集、并集等等，如第17-27行所示，有两种实现方式。

例程1-54

```

第1行 #set应用示例
第2行
第3行 aFruit={"Apple","Peach","Banana","Cherry","Ginkgo","Grape","lichee"}
第4行 bFruit={"Cherry","Ginkgo","Grape","lichee"}
第5行
第6行 print(aFruit)
第7行 print(bFruit)
第8行 print(aFruit.add("Apricot"))#增加成员
第9行 print(aFruit.remove("Ginkgo"))#删除成员
第10行 print(set("Apple"))#A、p(只有一个p)、l、e，无重复无序序列，每次顺序未必稳定
第11行
第12行 print("\n将list转换为集合")
第13行 cFruit=set(["Watermelon","Tangor","Pomelo"])#将list转换Set
第14行 print(cFruit)
第15行 print(len(cFruit))#返回cFruit数量
第16行
第17行 print("\n集合运算的运算符示例")
第18行 print(aFruit-bFruit)#求差集，项在aFruit但不在bFruit中
第19行 print(aFruit&bFruit)#求交集，项既在aFruit又在bFruit中

```

```

第20行 print(aFruit|bFruit)#求并集，对两个集合合并，并去重
第21行 print(aFruit^bFruit)#求对称差集，在aFruit仅bFruit中出现一次
第22行
第23行 print("\n集合运算的函数示例")
第24行 print(aFruit.difference(bFruit))#求差集，项在aFruit但不在bFruit中
第25行 print(aFruit.intersection(bFruit))#求交集，项既在aFruit又在bFruit中
第26行 print(aFruit.union(bFruit))#求并集，对两个集合合并，并去重
第27行 print(aFruit.symmetric_difference(bFruit))#求对称差集，在aFruit仅bFruit中出现一次
第28行
第29行 print("\n集合运算的函数版本支持序列类型，其他几个函数也支持")
第30行 print(aFruit.union(["Sugarcane","Mangosteen","Chestnum"]))#求对称差集，在aFruit仅bFruit中出现一次
第31行
第32行 print("\naFruit集合全部成员：\n",aFruit)
第33行 print(set(["Apple","Grape"]).issubset(aFruit))#set(["Apple","Grape"])是否aFruit的子集
第34行 print(aFruit.issuperset(set(["Apple","Grape"])))#aFruit是否包含set(["Apple","Grape"])
第35行
第36行 aFruit.clear()#清除字典中全部成员
第37行 bFruit.clear()
第38行 cFruit.clear()
第39行
第40行 #eof

```

## 5、Dict(字典序列)

Dict是Dictionary的简写，其功能也与Dictionary相似，即“词条-解释”关系，在程序设计中称之为“key-value”（键-值）。如同例程1-55第3-4行所示，Apple对应苹果、Peach对应桃等等。在其他语言中，Dict又称为Map(映射)，如C++。在Python的Dict类型中，可以必须唯一不能重复，如同身份证号对应姓名是唯一对应关系。注意：键与值之间用冒号间隔，每个键值对之间用逗号间隔。与Set类型相比，都是用花括号申明，**但Set有Key无值，而Dict则有Key有值。**

### 例程1-55

```

第1行 #Dict应用示例,Dict以花括号对界定
第2行
第3行 myDict={"Apple":"苹果","Peach":"桃","Banana":"香蕉",
第4行 "Cherry":"樱桃","Ginkgo":"银杏","Grape":"葡萄","lichee":"荔枝"}
第5行
第6行 print(myDict["Apple"]) #输出：苹果
第7行
第8行 print(myDict)#输出Dict全部内容，与原始序列顺序未必一致
第9行
第10行 print(myDict.keys())#输出全部Key值，类型为Set
第11行
第12行 print(myDict.values())#输出全部Value，类型为Set
第13行
第14行 for fruitENG in myDict:
第15行     print(fruitENG,"--->",myDict[fruitENG])
第16行
第17行 print()#输出一个空行
第18行

```

```

第19行 for (fruitENG,fruitCHN) in myDict.items():
第20行     print(fruitENG,"--->",fruitCHN)
第21行
第22行 print(myDict.items())
第23行 #输出为dict_items([('Apple', '苹果'), ('Grape', '葡萄'),....., ('Peach', '桃'), ('Banana', '香蕉')])
第24行
第25行 print(myDict["Apple"]);#输出: 苹果, 如果不存在将导致KeyError错误
第26行 print("Apple" in myDict);#输出: True
第27行 print(myDict.get("Apple",None));#输出: 苹果
第28行 print(myDict.get("apple",None));#输出: None
第29行
第30行 fruitName="Apple"
第31行 if fruitName in myDict:
第32行     print(myDict[fruitName])
第33行
第34行 herDict={"Lemon": "柠檬", "Mango": "芒果"}
第35行 myDict.update(herDict)
第36行 print(myDict)
第37行
第38行 myDict.clear();#清除字典中全部成员
第39行
第40行 #eof

```

Dict.keys()能以Set类型获得指定Dict的Key, 如第10行所示; Dict.values()能以Set类型获得值, 如第12行所示; Dict.items()能获得所有Key-Value对, 如第19-20行所示。

例程第25行代码myDict["Apple"]的功能是查找"Apple"对应的value(值), 如果"Apple"不存在或者输入错误(也常常是不存在), 将导致KeyError错误。为规避该错误, 可以用"Apple" in myDict方式, 先查找是否存在, 或者第27、28行模式, 如果不存在, 则返回get()第二个参数的值, 默认为None。

两个Dict类型不能相加合并成一个新的Dict, 但可以如例程第35行所示, 将herDict更新到myDict之中。例程1-56是Dict的一个应用示例。

## 例程1-56

```

第1行 #Dict应用示例, 统计字符出现次数
第2行
第3行 strA=""Over six years ago, in December 1989, I was looking for a "hobby"
第4行 programming project that would keep me occupied during the week around
第5行 Christmas. My office ... would be closed, but I had a home computer, and
第6行 not much else on my hands. I decided to write an interpreter for the new
第7行 scripting language I had been thinking about lately: a descendant of ABC
第8行 that would appeal to Unix/C hackers. I chose Python as a working title
第9行 for the project, being in a slightly irreverent mood (and a big fan of
第10行 Monty Python's Flying Circus).""
第11行
第12行 charStat={}
第13行 for singleChar in strA:
第14行     if singleChar not in charStat:
第15行

```

	charStat[singleChar]=1
第16行	else:
第17行	charStat[singleChar]+=1
第18行	
第19行	print(charStat)#输出每个字符出现次数
第20行	
第21行	#eof

## 第七节 序列函数

### 1、统计函数：max()、min()和sum()

Python提供了3个常用函数max()、min()和sum()，分别用于最大值、求最小值和求和，其中的max()和min()还支持命名参数key，用于对每个数据成员进行处理并在此基础上求得最大值或最小值，例程1-57是其应用示例。

例程1-57

第1行	lstA=[12,-12,45,-345,12,7,24,87]
第2行	
第3行	minVal=min(lstA)
第4行	print(minVal)#输出： -345
第5行	
第6行	maxVal=max(lstA)
第7行	print(maxVal)#输出： 87
第8行	
第9行	sumVal=sum(lstA)
第10行	print(sumVal)#输出： 170
第11行	
第12行	#min()、max()支持key参数
第13行	minVal=min(lstA,key=lambda x:abs(x))
第14行	print(minVal)#输出： 7
第15行	
第16行	maxVal=max(lstA,key=lambda x:abs(x))
第17行	print(maxVal)#输出： -345
第18行	
第19行	nameScore={"张珊":89,"李思":76,"王武":90,"刘柳":98}
第20行	
第21行	#nameScore.items()组成一个tuple序列
第22行	maxScore=max(nameScore.items(),key=lambda x:x[1])
第23行	print("最高分学生： {}({}分)".format(maxScore[0],maxScore[1]))
第24行	
第25行	#sum()不支持关键字key
第26行	sumScore=sum([nameScore[i] for i in nameScore])
第27行	print("总分： ",sumScore)
第28行	
第29行	#eof

### 2、过滤函数：filter()

filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素，如例程1-58所示。

```

第1行 #filter()函数的使用
第2行 def isOdd(x):#判断是否奇数
第3行     return x%2==1
第4行
第5行 listA=range(1,100)
第6行 listOdd=list(filter(isOdd,listA))#过滤掉偶数
第7行 #输出过滤后结果
第8行 print(listOdd)#输出：全部是奇数
第9行
第10行 listB=range(1,100)
第11行 listEven=list(filter(lambda x:x%2==0,listB))#过滤掉奇数
第12行 #输出过滤后结果
第13行 print(listEven)#输出：全部是偶数
第14行
第15行 #eof

```

### 3、映射函数：map()

映射函数用于将一个或多个原序列按照某种指定规则转为另外一个序列，返回值类型为map，可以通过list()等函数转换为指定类型。例程1-59中listB通过lambda函数给listB的每个成员加10，然后通过list()函数将其转换为list类型。

```

第1行 #map()函数的使用
第2行
第3行 listB=range(1,10)
第4行 listMapA=list(map(lambda x:x+10,listB))#listB中每一个数都加上10
第5行 print(listMapA)
第6行
第7行 def numChange(N):#偶数乘以2，否则加上2
第8行     if N%2==0:
第9行         return N*2
第10行     else:
第11行         return N+2
第12行
第13行 listMapB=list(map(numChange,listB))#listB中每一个数都加上10
第14行 print(listMapB)
第15行
第16行 listA=[1,2,3,4]
第17行 listB=[5,6,7,8]
第18行 print(list(map(lambda x,y:x+y,listA,listB)))
第19行
第20行 #eof

```

map()函数支持多个序列，如例程1-59第16-18行所示，此时要求函数也是多参数，与序列的个数保持一致，每个序列与序列之间保持一一对应关系，即第1个序列的第1个值与第2个序列第1个值对应。如果某个序列较长，则会按照短序列数量截断。

### 4、排序函数：sorted()



sorted()函数应用非常广泛，返回值为list类型，其有三个参数，第1个是指定被排序序列，第2个是指定排序函数，第3个指定是降序还是升序，其中第2个和第3个参数可以省略，当省略直接按序列成员值进行排序，同时按升序排列。第4行sorted()函数仅有1个参数，指定被排序序列，第8行则有2个参数，第15行则有3个参数。

## 例程1-60

```
第1行 #sorted()函数的使用
第2行
第3行 listA=[1,2,-10,5,12,30,-9]
第4行 listB=sorted(listA)
第5行 print(listB)#输出: [-10, -9, 1, 2, 5, 12, 30]
第6行
第7行 #可以指定排序函数，此处按绝对值
第8行 print(sorted(listA,key=abs))
第9行
第10行 listC=range(1,50)
第11行 listD=sorted(listC,key=lambda x:x%3)#按3的余数排列
第12行 print(listD)
第13行
第14行 print(sorted(range(1,20),key=lambda x:x%2))#偶数在前，奇数在后
第15行 print(sorted(range(1,20),key=lambda x:x%2,reverse=True))#奇数在前，偶数在后
第16行
第17行 listE=['apple','banana','strawberry','Chery']
第18行 print(sorted(listE))
第19行 print(sorted(listE,key=str.lower))
第20行 print(sorted(range(1,1000),key=str))#按字符串排序
第21行
第22行 def Mod10(N):
第23行     return N%10
第24行
第25行 listF=[11,10,3,78,21,43,25]
第26行 print(sorted(listF,key=Mod10))#输出: [10, 11, 21, 3, 43, 25, 78]
第27行
第28行 #eof
```

sorted()函数同样可以用于字典排序，如例程1-61所示。

## 例程1-61

```
第1行 #字典排序
第2行 mapA={"西藏":122,"新疆":166,"四川":48,"青海":72,"内蒙":118}
第3行 listA=sorted(mapA.items(),key=lambda x:x[1])
第4行 print(listA)
第5行 #[(('四川', 48), ('青海', 72), ('内蒙', 118), ('西藏', 122), ('新疆', 166))]
第6行
第7行 #按key排序
第8行 listC=sorted(mapA.keys())
第9行 print(listC) #按汉字的编码排序，顺序未必与直观符合
第10行 #['内蒙', '四川', '新疆', '西藏', '青海']
```

第11行

第12行 #可以value排序

第13行 listD=sorted(mapA.values())

第14行 print(listD) #[48, 72, 118, 122, 166]

第15行

第16行 #eof

## 5、组合函数: zip()

zip()函数用于将多个序列数据组合新的序列, 如例程1-62所示。

### 例程1-62

第1行 #zip()函数的使用示例

第2行

第3行 X={"张山","李斯","王武"}

第4行 Y=[100,90,65]

第5行 Z=[90,70,80]

第6行

第7行 xyz=zip(X,Y,Z)

第8行 print(type(xyz))#输出: <class 'zip'>

第9行 for i in xyz:

第10行 print(i,end="\$")

第11行 #上行输出: ('王武', 100, 90)\$\$('张山', 90, 70)\$\$('李斯', 65, 80)\$

第12行

第13行 #两个不等长序列组合, 长序列将被部分舍去

第14行 a=[1,2,3]

第15行 b=[4,5,6,7]

第16行 ab=zip(a,b)

第17行 for i in ab:

第18行 print(i,end="\$")#输出: (1, 4)\$\$(2, 5)\$\$(3, 6)\$

第19行

第20行 #单序列组合

第21行 c=[1,2,3]

第22行 cZip=zip(c)

第23行 for i in cZip:

第24行 print(i,end="\$")#输出: (1,)\$\$(2,)\$\$(3,)\$

第25行

第26行 #空序列组合

第27行 d=[]

第28行 dZip=zip(d)

第29行 for i in dZip:

第30行 print(i,end="\$")#输出: 没有输出, 空序列组合没有内容

第31行

第32行 #eof

## 6、反序函数: reversed()

reversed()函数用于将指定序列前后颠倒, 如例程1-63所示。

### 例程1-63

```

第1行 #reversed()函数的使用示例
第2行 listA=[12,23,34,45,56]
第3行 reversedlistA=reversed(listA)
第4行 print(type(reversedlistA))#输出: <class 'list_reverseiterator'>
第5行 for i in reversedlistA:
第6行     print(i,end="$")#输出: 56$45$34$23$12$
第7行
第8行 #等价实现, 会显得麻烦
第9行 for i in range(-1,-len(listA)-1,-1):
第10行     print(listA[i],end="$")#输出: 56$45$34$23$12$
第11行
第12行 strB="abcXYZ"
第13行 strC=reversed(strB)
第14行 for i in strC:
第15行     print(i,end="$")#输出: Z$Y$X$c$b$a$
第16行
第17行 #eof

```

## 第八节 列表解析

例程1-64第3行代码即为列表解析, 其功能是产生一个list。列表解析的执行过程是: 在for-in循环中, 依次取出序列中的成员, 并把成员的值赋给循环变量, 并执行for-in循环左侧的表达式。列表解析的功能完全可用循环实现, 但列表解析通常速度快(据测试, 一般快一倍), 且代码简洁。比较第6-8行与第3行即可明显感知。

列表解析可以适用于str、list、Dict、Set等等支持for-in循环的序列类型。

例程1-64

```

第1行 #! /usr/bin/python
第2行
第3行 listA=[x+10 for x in range(10)]
第4行 print(listA)#输出: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
第5行
第6行 listB=[]
第7行 for x in range(10):
第8行     listB.append(x+10)
第9行 print(listB)#输出: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
第10行
第11行 strA="ABCDEFGFG"
第12行 listC=[c*2 for c in strA]
第13行 print(listC)#输出: ['AA', 'BB', 'CC', 'DD', 'EE', 'FF', 'GG']
第14行
第15行 #产生偶数list
第16行 listEven=[i for i in range(0,10) if i%2==0] #过滤, 进留下偶数
第17行 print(listEven)
第18行
第19行 listA=["boy","code","quick","create","Python","Ruby","algorithm"]
第20行 print([word for word in listA if len(word)>4]) #过滤掉listA中长度小于等于4的条目
第21行

```

```

第22行 #建立每个单词对应长度的dict
第23行 wordDict={word:len(word) for word in listA}
第24行 print(wordDict)
第25行
第26行 #eof

```

## 例程1-65

```

第1行 #三种列表生成模式
第2行 def isPrime(N):
第3行     if N<=1:return False
第4行     if N==2:return True
第5行     for i in range(2,N):
第6行         if N%i==0:return False
第7行     return True
第8行
第9行 rangeData=range(2,10000)
第10行
第11行 listPrime=[]
第12行 for i in rangeData:
第13行     if isPrime(i)==True:listPrime.append(i) #循环生成质数列表
第14行
第15行 listPrime=[i for i in rangeData if isPrime(i)==True] #列表解析生成质数列表
第16行
第17行 listPrime=list(filter(isPrime,rangeData)) #过滤函数生成质数列表
第18行
第19行 #eof

```

例程1-66是对基本列表解析的扩展。第3行代码之后增加了if子句，是对for-in循环成员的筛选或过滤；第9行代码是两个for-in循环，Python的列表解析支持多个for-in循环；第12行的每个for-in循环增加了if子句，Python的列表解析支持多个for-in循环，每个for-in循环都可以含有if子句。

## 例程1-66

```

第1行 #! /usr/bin/python
第2行
第3行 listA=[x+10 for x in range(10) if x%3==0]
第4行 print(listA)#print:[10, 13, 16, 19]
第5行
第6行 listB=[1,2]
第7行 listC=[11,12,13]
第8行
第9行 listD=[x+y for x in listB for y in listC]
第10行 print(listD)#print:[12, 13, 14, 13, 14, 15]
第11行
第12行 listE=[x*y for x in range(1,10) if x%3==0 for y in range(10,20) if y%5==0]
第13行 print(listE)#print:[30, 45, 60, 90, 90, 135]
第14行

```

## 第九节 基本文件操作

文件是重要的输入输出，能实现数据持久。对文件的操作，可以分为读(reading)和写(writing)。例程1-67是文件的写操作，将内存中的数据写入到文件之中，使得数据可以持久保存，而不是随着程序运行结束而消失。文件操作首先是打开文件，在Python中用open()函数实现文件打开，打开时需要指定文件是用于读还是写，如例程第3行所示，open()函数打开一个名为“test.txt”的文件，第2个参数的值为“w”，表明该文件用于写，即程序向指定的文件此处为test.txt写入数据。如果文件用于写，则指定第2个参数为“r”即可。open()函数执行完毕后，将生成一个对象，在例程第3行，该对象名称为writingToFile(当然也可以是其他合规的名称)。文件成功打开后，就可以对文件进行读或写操作，例程第4行的write()用于向文件写入数据，为文本类型文件。当文件操作完毕后，一定要关闭，第5行的close()即用于关闭文件，停止程序与文件之间的联系。

**例程1-67**

```
第1行  #文件操作---写
第2行
第3行  writingToFile=open('test.txt','w')
第4行  writingToFile.write('Hello!Python!')
第5行  writingToFile.close()
第6行
第7行  #eof
```

write()只能将str类型的数据写入文件，如果要写入其他类型的数据，必须进行类型转换，如例程1-68第6行所示，此处将int转换为str。写入文件时，write()将不自动在两个数据之间添加空格，将第6行代码改成wFile.write(str(i)+' ')即可在每个数据之间增加空格。

**例程1-68**

```
第1行  #文件操作---写
第2行
第3行  wFile=open('test.txt','w')
第4行
第5行  for i in range(1,20):
第6行      wFile.write(str(i))#write()的参数必须为str数据类型
第7行      #在文件中存储为12345678910111213141516171819
第8行
第9行  wFile.close()
第10行
第11行  #eof
```

除write()外，print()亦可实现文件输出，如例程1-69所示。

**例程1-69**

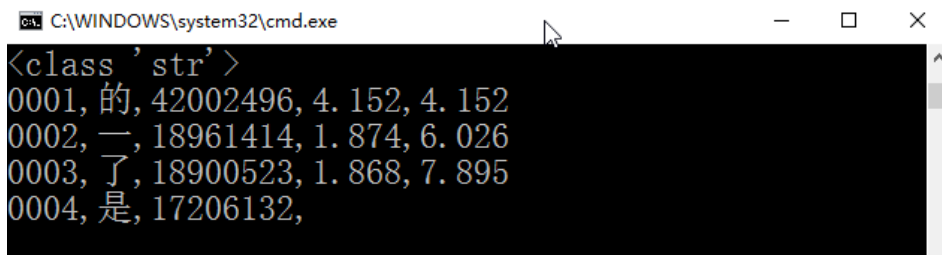
```
第1行  #文件操作---写
第2行
第3行  wFile=open('test.txt','w')
第4行
第5行  for i in range(1,20):
第6行      print(i,file=wFile)#默认每个数之后换行
第7行
第8行  wFile.close()
第9行
第10行
```

```
#eof
```

文件可以写，也可以读，例程1-70即用于将一个名为6763HZ.txt的文本文件读入到程序的txtHZ6763变量之中。该文件的内容如图1-6所示。图中顶部的<class 'str'>是例程第9行执行结果，即读入内容讲义Str类型存在。第11行执行结果如图中最后4行所示。6763HZ.txt中的内容是6763个汉字字频统计，第1列是序号，第2列是汉字，第3列是汉字在样本中出现次数，第3列是该汉字占比，第4列是累积比例。

#### 例程1-70

```
第1行  #! /usr/bin/python
第2行
第3行  #文件操作：读取文本文件
第4行
第5行  fRead=open("6763HZ.txt","r") #打开文件
第6行  txtHZ6763=fRead.read() #读入数据，一次性全部读完
第7行  fRead.close() #关闭文件
第8行
第9行  print(type(txtHZ6763)) #检测读出内容的类型
第10行
第11行  print(txtHZ6763[0:100])
第12行
第13行  #eof
```



```
C:\WINDOWS\system32\cmd.exe
<class 'str'>
0001, 的, 42002496, 4. 152, 4. 152
0002, 一, 18961414, 1. 874, 6. 026
0003, 了, 18900523, 1. 868, 7. 895
0004, 是, 17206132,
```

图1-10 例程1-70执行效果图

Python中的read()是将文件的中内容一次全部读入内存，如果文件很大，就有可能超过内存容量，导致程序不能运行。另外，整体一次读入不便于逐行处理数据。Python提供逐行读入，如例程1-71所示，其功能是将6763HZ.txt逐行读入，第13行执行后显示效果如图1-6所示。

#### 例程1-71

```
第1行  #! /usr/bin/python
第2行
第3行  #文件操作：读取文本文件
第4行
第5行  fRead=open("6763HZ.txt","r") #打开文件
第6行
第7行  dataFromFile=[] #空list
第8行  for line in fRead:
第9行      dataFromFile.append(line) #新读入每一行追加到dataFromFile之中
第10行
第11行  fRead.close() #关闭文件
第12行
第13行  print(dataFromFile[0:3])
第14行
```

```
C:\WINDOWS\system32\cmd.exe
D:\myBook\Python\Python_Prog>python 2017022600.py
['0001, 的, 42002496, 4. 152, 4. 152\n', '0002, 一, 18961414, 1. 874, 6. 026\n', '0003, 了, 18900523, 1. 868, 7. 895\n']
```

图1-11 例程1-71执行效果

逐行读入还可以用`readline()`和`readlines()`方式，如例程1-72采用`readline()`方式，其效果与例程1-71相同。

## 例程1-72

```

第1行  #! /usr/bin/python
第2行
第3行  #文件操作：读取文本文件
第4行
第5行  fRead=open("6763HZ.txt","r") #打开文件
第6行
第7行  dataFromFile=[] #空list
第8行  readingline=fRead.readline() #第一次读入
第9行  while readingline: #当到达文件尾时，readingline为False
第10行      dataFromFile.append(readingline) #新读入每一行追加到dataFromFile之中
第11行      readingline=fRead.readline() #继续读入
第12行
第13行  fRead.close() #关闭文件
第14行
第15行  print(dataFromFile[0:3])
第16行
第17行  #eof

```

`readlines()`与`read()`一样能一次性将全部数据读入，但读入类型不是整体为`Str`，而是逐行读入其以每行为成员的`list`类型，如例程1-73所示。很明显，如果是全部读入，`readlines()`有很好的性能优势。

## 例程1-73

```

第1行  #! /usr/bin/python
第2行
第3行  #文件操作：读取文本文件
第4行
第5行  fRead=open("6763HZ.txt","r") #打开文件
第6行
第7行  dataFromFile=[] #空list
第8行  dataFromFile=fRead.readlines() #一次性读入
第9行
第10行 fRead.close() #关闭文件
第11行
第12行 print(dataFromFile[0:3])
第13行
第14行 #eof

```

上述例程并不能方便地查找每个汉字的数据，例程1-74是对上述例程的改造。当输入汉字时，能立即显示该汉字对应的数据。



```

第1行  #! /usr/bin/python
第2行
第3行  #文件操作：读取文本文件
第4行
第5行  fRead=open("6763HZ.txt","r") #打开文件
第6行
第7行  dataFromFile=[] #空list
第8行  dataFromFile=fRead.readlines() #一次性读入
第9行
第10行 fRead.close() #关闭文件
第11行
第12行 dictHZ={}
第13行 arrline=()
第14行 for line in dataFromFile:
第15行     arrline=line.split(",") #将每行切分为Tuple
第16行     dictHZ[arrline[1]]=(int(arrline[0]),int(arrline[2]),float(arrline[3]),float(arrline[4]));
第17行
第18行 hzInput=input("请输入单个汉字：")
第19行 hzData=dictHZ[hzInput]
第20行 print("\n\n汉字:",hzInput,"\n序号：",hzData[0],"出现次数：",hzData[1],"占比：",hzData[2],"累积占比：",hzData
第21行 [3])
第22行 #eof

```

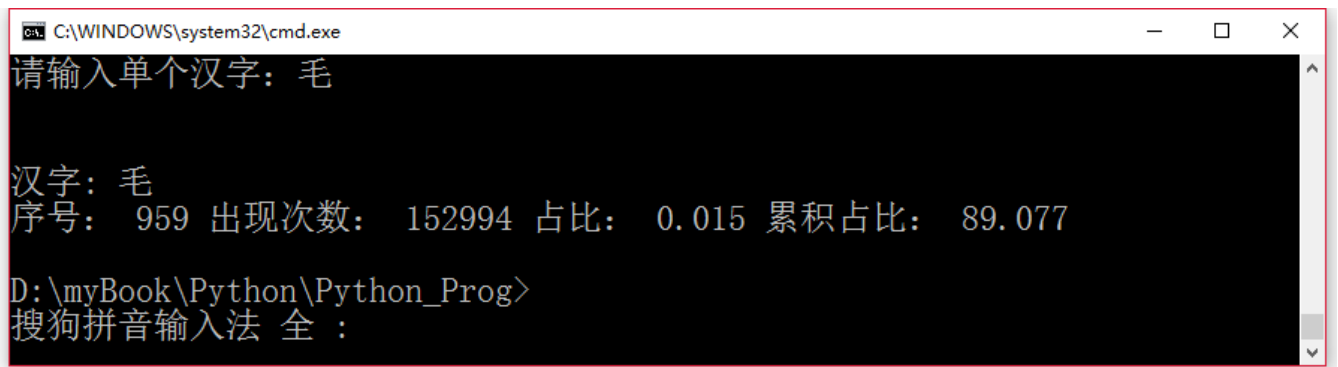


图1-12 例程1-74执行效果

打开文件函数open()的参数除了文件名、打开模式俩参数，常用的参数还有encoding和errors。encoding用于设置被打开文件的编码，比如：utf-8、gb2312等等；errors用于设置文件读取时错误处理策略。如例程1-75所示。在本例程中，如果输入1，将得到每个汉字在红楼梦中出现次数。由于被处理文件“红楼梦.txt”是utf-8编码，不是操作系统默认编码，因此必须设定编码类型。由于该文档来自网络下载，包含一些编码不合规字符，处理时可能发生错误。在中文字频统计中，个别字符出现问题并不影响宏观，因此设置errors参数值为ignore，即忽略该错误继续处理，否则将导致程序运行异常。errors参数值还可以设置为strict、replace等等。

```

第1行  #! /usr/bin/python
第2行
第3行  #红楼梦N元组合程序
第4行

```

```

第5行 readingFromFile= open('红楼梦.txt','r',encoding="utf-8",errors="ignore")
第6行
第7行 contentFromFile=readingFromFile.read()
第8行 readingFromFile.close()
第9行
第10行 charStat={}#定义charStat为dict类型
第11行 contentLength=len(contentFromFile)#获得字符数量
第12行 word=""
第13行 N=int(input("请输入N元的N值，必须为整数请小于7："))
第14行 for i in range(contentLength-N):
第15行     word=contentFromFile[i:i+N]
第16行     if word not in charStat:
第17行         charStat[word]=1
第18行     else:
第19行         charStat[word]+=1
第20行
第21行 sortedCharDict=sorted(charStat.items(),key=lambda d:d[1],reverse=True)
第22行
第23行 wFile=open('item.txt','w',encoding="utf-8",errors="ignore")
第24行 for wordItem in sortedCharDict:
第25行     print(wordItem[0],"\t",wordItem[1],file=wFile)
第26行
第27行 #eof

```

计算机的基础是二进制，内存是二进制，外存如硬盘U盘等等都是二进制。Python提供把内存中的变量对象等直接存储到外存文件中，或直接从外存读入放到变量或对象之中的标准库pickle。内存是的存放是暂时，程序运行结束内存释放，保存到文件，实现了内存内容持久化。如例程1-76所示。

#### 例程1-76

```

第1行 import pickle
第2行
第3行 listName=["贾宝玉","林黛玉","薛宝钗","贾元春","贾迎春","贾探春","贾惜春"]
第4行 outFile=open(r"D:\listName.pkl","wb") #以二进制方式打开文件
第5行 pickle.dump(listName,outFile) #将listName输出到outFile之中
第6行 outFile.close()
第7行
第8行 del listName #删除listName，释放listName占据的内存空间
第9行
第10行 inFile=open(r"D:\listName.pkl","rb") #以二进制读方式打开文件
第11行 nameList=pickle.load(inFile) #load是pickle对象的方法
第12行 for Name in nameList:
第13行     print(Name,end="\t")
第14行
第15行 #eof

```

可以将多个变量组合成一个变量，然后用pickle存储到文件，如例程1-77所示。

#### 例程1-77

```

第1行 import pickle
第2行
第3行 listNameA=["贾宝玉","林黛玉","薛宝钗","贾元春","贾迎春","贾探春","贾惜春"]
第4行 listNameB=["刘备","诸葛亮","曹操","孙权","关羽","张飞","赵子龙"]
第5行 listC=[listNameA,listNameB] #listC是list的list
第6行
第7行 outFile=open(r"D:\listList.pkl","wb") #以二进制方式打开文件
第8行 pickle.dump(listC,outFile) #将listName输出到outFile之中
第9行 outFile.close()
第10行
第11行 del listC #删除listName, 释放listName占据的内存空间
第12行 del listNameA
第13行 del listNameB
第14行
第15行 inFile=open(r"D:\listList.pkl","rb") #以二进制读方式打开文件
第16行 nameList=pickle.load(inFile) #load是pickle对象的方法
第17行 inFile.close()
第18行
第19行 print(nameList)
第20行
第21行 #eof

```

Python还提供一个shelve标准库, 能把变量或对象以字典形式存储到文件之中, 如例程1-78所示, 多于多变量或者多对象, 这种方式较pickle更加灵活。

#### 例程1-78

```

第1行 import shelve
第2行
第3行 listNameA=["贾宝玉","林黛玉","薛宝钗","贾元春","贾迎春","贾探春","贾惜春"]
第4行 listNameB=["刘备","诸葛亮","曹操","孙权","关羽","张飞","赵子龙"]
第5行 dictKongFu={"郭靖":"降龙十八掌","黄药师":"弹指神通","段誉":"六脉神剑"}
第6行
第7行 dictSlv=shelve.open(r"dictData.slv")#dictSlv相当于一个空字典
第8行
第9行 dictSlv["nameA"]=listNameA
第10行 dictSlv["nameB"]=listNameB
第11行 dictSlv["KongFu"]=dictKongFu
第12行 dictSlv.close() #数据已经存放到文件
第13行
第14行 del listNameA
第15行 del listNameB
第16行 del dictKongFu
第17行 del dictSlv
第18行
第19行 dictData=shelve.open(r"dictData.slv")
第20行 print(dictData["nameA"])

```

第21行	<code>print(dictData["nameB"])</code>
第22行	<code>print(dictData["KongFu"])</code>
第23行	
第24行	<code>dictData.close()</code>
第25行	
第26行	<code>#eof</code>

除了文本文件读写外，还有其他操作，如二进制文件读写，如图像等，更多文件操作，请参考后续相关章节。

**第十节 小结**