第三章　数据组织

本章导读

一名多量有多种类型，已学过list和tuple，本章将学习集合set和字典dict。不同类型，有不同特征，执行效率或易用程度不尽相同，也就有了不同应用场景。

将数据根据需要存储在不同类型之中，以更好地满足不同应用，在其他语言中称之为数据结构（Data Structure）。在Python中，一些基本的数据结构已经集成在语言之中，以更方便程序设计，达到数据组织。

在Python中，list和tuple是有序数据的集合即维持数据的先后顺序，但tuple的数据一旦生成，不能变化。集合set和dict不维持数据先后顺序，且数据不能重复，但能增加删除修改数据。dict的每个成员由两个数据组成称之为键值对（key-value pair），每个成员的键不能重复，必须唯一。set则只有键，没有对应的值。

本章不仅将学习set和dict，还将比较不同类型的特征，类型转换以及相关的处理函数。

学习目标：

1. 掌握set和dict的特征；
2. 掌握list、tuple、set和dict的异同；
3. 掌握不同类型之间的转换；
4. 掌握列表解析；
5. 掌握相关处理函数等；

本章目录

第一节　字典结构
第二节　集合结构
第三节　异同比较
第四节　列表解析
第五节　类型转换
第六节　数据函数
　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()
第七节　题目示例

第一节　字典结构

dict字典结构应用广泛，凡“一对一”且要频繁使用查找删改等几乎都可以使用dict。dict是Dictionary的简写，其功能也与Dictionary相似，即一个词条对应一条解释的“词条-解释”关系，在程序设计中称之为“key-value pair”(键-值对)。如同例程3-1第3行所示，Apple对应苹果、Peach对应桃等。在其他语言中，dict又称为Map(映射)，如C++。在Python的dict类型中，key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：**dict采用花括号，键与值之间用冒号间隔，每个键值对之间用逗号间隔。**

例程3-1

```
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","Cherry":"樱桃","Grape":"葡萄"}
第4行  print(myDict["Apple"]) #输出：苹果
```

```
第5行  print(myDict)#输出Dict全部内容
第6行
第7行  myDict["Ginkgo"]="银杏" #增加新成员
第8行  myDict["Cherry"]="车厘子" #修改成员对应值
第9行
第10行 print("Grape" in myDict) #输出：True
第11行 print("葡萄" in myDict) #输出：False
第12行 print(myDict)
```

对于组合数据类型而言，成员访问功能很常用，对list、tuple、str都是通过方括号运算符，dict也不例外。不过，字典使用key而不是序号，如例程第4行myDict["Apple"]，Apple是key。**如果key不存在，将导致KeyError。**

增加新成员的方式如例程3-1第7行所示，如果方括号中的key不存在则增加新成员，如果已经存在则将修改成员的值，如例程第8行所示。

成员运算符in和not in同样适用dict，如例程3-1第10行所示，如果Grape存在于myDict之中则返回值为True，在例程中返回True。**注意，in和not in都是用key判断**，而不是value，所以例程第11行就是False。

例程3-2是not in在dict中的应用示例，第14行代码c not in charStat是判断变量c是否存在于charStat之中。如果不存在返回True，否则False。第15行代码将在c不存在于charStat时执行，即当c所代表的字符尚未存在于charStat时执行，也就是第一次循环到某个字符时执行，此时该字符对应的出现次数是1，当已经存在时，则累加1（第17行）。经过循环后，将统计出每个字符的出现次数。

例程3-2

```
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={} #存储每个字符出现次数，形如{'e':44,'r':25,'a':30}
第13行 for c in strA:
第14行     if c not in charStat:
第15行         charStat[c]=1
第16行     else:
第17行         charStat[c]+=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
```

dict的函数keys()、values()、items()可以分别获得全部成员的键key、值value以及键值对key-value，如例程3-3所示。从图3-1可以看出，第1-3行的输出，不是单个值，但也不是方括号list、圆括号tuple、花括号dict也不是引号界定的字符串。类似形式，一般说来是类对象自有数据类型，但通常都支持for-in循环这种最常用的处理方法。如果不能确定是否支持，可以通过for-in循环测试，如果不报错，就表示支持。通过本例中的第7、8行可以看出dict.items()函数的返回值支持for-in循环，dict.keys()和dict.values()都支持for-in循环。

例程3-3

```
第1行  #Dict应用示例,获得dict的数据，全部key、全部value、所有条目items等。
第2行
第3行  myDict={"Apple":"苹果","Peach":"桃","Banana":"香蕉","Cherry":"樱桃","Grape":"葡萄"}
第4行  print(myDict.keys()) #获得全部key
第5行  print(myDict.values()) #获得全部value
第6行  print(myDict.items()) #获得全部key-value
第7行
第8行  for everyPair in myDict.items(): #通过for-in循环
第9行      print(everyPair)
第10行
第11行 for (eng,chn) in myDict.items():
第12行     print(f"{eng}-->{chn}",end=",")
```

```
dict_keys(['Apple', 'Peach', 'Banana', 'Cherry', 'Grape'])
dict_values(['苹果', '桃', '香蕉', '樱桃', '葡萄'])
dict_items([('Apple', '苹果'), ('Peach', '桃'), ('Banana', '香蕉
'), ('Cherry', '樱桃'), ('Grape', '葡萄')])
('Apple', '苹果')
('Peach', '桃')
('Banana', '香蕉')
('Cherry', '樱桃')
('Grape', '葡萄')
Apple-->苹果,Peach-->桃,Banana-->香蕉,Cherry-->樱桃,Grape-->葡萄,
```

图3-1　例程3-1的执行效果

从程序可以看出，dict.items()是由key-value构成的tuple，可以用tuple对应之，获得每一个值，有时能更加方便地工作，如例程第11、12行所示。dict.items()形成tuple将一一对应到(eng,chn)。

dict类型的数据可以转换为其他数据类型，如例程3-4所示，图3-2是其执行效果。第3-6行是给变量myDict赋值，内容为每个英文字母的出现频率。第8行是将myDict转换为list，从图中可以看出，输出内容确实是将dict类变量myDict的key转换成了方括号类型的list。第11、12行是将myDict转换为tuple，第14、15行是将myDict.items()转换为list，从图中可以看出，list的每个成员为tuple，第1个值为key，第2个值为value。转换为list后，可以利用list类对象的排序函数sort()，如例程第17行所示。

例程3-4

```
第1行  #Dict应用示例,dict与其他类型之间的转换
第2行
第3行  myDict={"a":0.08167,"b":0.01492,"c":0.02782,"d":0.04253,"e":0.12702,"f":0.02228,"g":0.02015,
第4行          "h":0.06094,"i":0.06966,"j":0.00153,"k":0.00772,"l":0.04025,"m":0.02406,"n":0.06749,
第5行          "o":0.07507,"p":0.01929,"q":0.00095,"r":0.05987,"s":0.06327,"t":0.09056,"u":0.02758,
第6行          "v":0.00978,"w":0.02360,"x":0.00150,"y":0.01974,"z":0.00074}
第7行
第8行  toList=list(myDict)
第9行  print(toList)
第10行
第11行 toTuple=tuple(myDict.values())
第12行 print(toTuple)
第13行
第14行 toList=list(myDict.items())
第15行 print(toList)
```

```
第16行
第17行  toList.sort(key=lambda x:x[1],reverse=True)
第18行  print(toList)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', '
o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
(0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094,
0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749, 0.07507, 0.01929,
0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978, 0.0236, 0.0015, 0.
01974, 0.00074)
[('a', 0.08167), ('b', 0.01492), ('c', 0.02782), ('d', 0.04253), ('e', 0
.12702), ('f', 0.02228), ('g', 0.02015), ('h', 0.06094), ('i', 0.06966),
('j', 0.00153), ('k', 0.00772), ('l', 0.04025), ('m', 0.02406), ('n', 0.
06749), ('o', 0.07507), ('p', 0.01929), ('q', 0.00095), ('r', 0.05987),
('s', 0.06327), ('t', 0.09056), ('u', 0.02758), ('v', 0.00978), ('w', 0.
0236), ('x', 0.0015), ('y', 0.01974), ('z', 0.00074)]
[('e', 0.12702), ('t', 0.09056), ('a', 0.08167), ('o', 0.07507), ('i', 0
.06966), ('n', 0.06749), ('s', 0.06327), ('h', 0.06094), ('r', 0.05987),
('d', 0.04253), ('l', 0.04025), ('c', 0.02782), ('u', 0.02758), ('m', 0.
02406), ('w', 0.0236), ('f', 0.02228), ('g', 0.02015), ('y', 0.01974), (
'p', 0.01929), ('b', 0.01492), ('v', 0.00978), ('k', 0.00772), ('j', 0.0
0153), ('x', 0.0015), ('q', 0.00095), ('z', 0.00074)]
```

图3-2　例程3-4的执行效果

dict应用广泛。例程3-5用于统计《红楼梦》每个字出现的频率（包括标点符号），和前面几个例题有些相似。按照文件操作的常规套路，例程第1行打开文件，第2行读入文件内容，第3行关闭文件。第5行定义一个空的字典hzRate。第6-10行，逐一读入字符，如果该字符已经存在则累加1（第10行），否则则该字符的频率为1。第12行是将字典数据转换为list，然后在第13行按降序对数据排序。由于数据太多，此处仅输入前50项。如果将四大名著的字频放在一起对比，想必会有有趣结论。

例程3-5

```
第1行  hlmFile=open(r"c:\PythonCode\hlm.txt","r",encoding="utf-8")
第2行  hlmData=hlmFile.read()
第3行  hlmFile.close() #关闭文件
第4行
第5行  hzRate={} #用于保存每个字符（主要是汉字）的出现频率（出现次数）,以每个汉字为key。
第6行  for cn in hlmData:
第7行      if cn not in hzRate:
第8行          hzRate[cn]=1
第9行      else:
第10行         hzRate[cn]+=1
第11行
第12行 hzRateList=list(hzRate.items())
第13行 hzRateList.sort(key=lambda x:x[1],reverse=True)
第14行 print(hzRateList[:50]) #由于太多，直接print(hzRate)难以观察，此处仅输出前50例
```

```
[(', ', 57150), ('。', 28250), ('了', 20601), ('的', 15179), ('不', 145
73), ('一', 11758), (': ', 11355), ('来', 11109), ('道', 10716), ('“',
10658), ('人', 10253), ('是', 9823), ('说', 9454), ('我', 8895), ('”',
7714), ('这', 7576), ('他', 7518), ('你', 6956), ('去', 6056), ('着', 6
022), ('儿', 5928), ('也', 5868), ('玉', 5843), ('有', 5768), ('宝', 56
35), ('个', 5456), ('子', 5301), ('又', 5070), ('贾', 5048), ('里', 501
7), ('们', 4760), ('那', 4757), ('见', 4666), ('只', 4537), ('"', 4504)
, ('太', 4210), ('便', 3948), ('在', 3909), ('好', 3903), ('笑', 3802),
('家', 3800), ('上', 3708), ('么', 3588), ('得', 3479), ('姐', 3364), (
'大', 3347), ('头', 3321), ('听', 3201), ('就', 3177), ('出', 3129)]
```

图3-3　例程3-5的执行效果

和list、tuple等相似，字典也可以嵌套，如例程3-6所示，第4行的“小青龙汤”是字典键，其值为冒号后的8种成分与分量构成的字典，也就是值为字典。当然，key对应的值还可以是多种，如tuple、list、str等。第11行的tcm["小青龙汤"]将得到“小青龙汤”的值，该值为字典，仍然可以通过其键访问对应的值，所以print(tcm["小青龙汤"]["芍药"])的输出值为9。

例程3-6

```
第1行  #字典嵌套。仅为示例，不保证汤剂组成分量正确。
第2行
第3行  tcm={
第4行        "小青龙汤":{"麻黄":9,"芍药":9,"细辛":6,"干姜":6,"甘草":6,"桂枝":9,"五味子":6,"半夏":9},
第5行        "小柴胡汤":{"柴胡":3,"人参":2,"黄芩":2,"半夏":1.5,"甘草":1},
第6行        "大承气汤":{"大黄":12,"厚朴":24,"枳实":12,"芒硝":9},
第7行        "补中益气汤":{"黄芪":15,"人参":15,"白术":10,"炙甘草":15,"当归":10,"陈皮":6,
第8行        "升麻":6,"柴胡":12,"生姜":45,"大枣":24}
第9行      }
第10行
第11行 print(tcm["小青龙汤"]["芍药"])
第12行 for (Name,Element) in tcm.items():
第13行    print(Name,end="\n\t") #输出汤剂名称
第14行    for (eleName,itemWeight) in Element.items():
第15行       print(f"{eleName}:{itemWeight}",end=" ") #输出汤剂组成
第16行
第17行    print() #每个汤剂后换行
```

```
9
小青龙汤
        麻黄:9  芍药:9  细辛:6  干姜:6  甘草:6  桂枝:9  五味子:6  半夏:9
小柴胡汤
        柴胡:3  人参:2  黄芩:2  半夏:1.5  甘草:1
大承气汤
        大黄:12  厚朴:24  枳实:12  芒硝:9
补中益气汤
        黄芪:15  人参:15  白术:10  炙甘草:15  当归:10  陈皮:6  升麻:6  柴
胡:12  生姜:45  大枣:24
```

图3-4　例程3-6的执行效果

对于dict，除了上述常用操作外，还有一些类里函数，如例程3-7。

例程3-7

```
第1行  #了解字典更多
第2行
```

```
第3行 aFruit={"Apple":"苹果","Peach":"桃","Banana":"香蕉","Cherry":"樱桃","Grape":"葡萄"}
第4行 bFruit={"Apple":"平安果","Pear":"梨","Orange":"橙","Lemon":"柠檬"}
第5行 cFruit={"Apple":"蛇果","Plum":"李子","mango":"芒果"}
第6行 dFruit=[("watermelon","西瓜"),("strawberry","草莓")]
第7行
第8行 dFruit=dict(dFruit)
第9行 print(dFruit) #将tuple转换为字典，输出：{'watermelon': '西瓜', 'strawberry': '草莓'}
第10行 cFruit.clear() #清除cFruit所有成员，cFruit成为空字典。
第11行 print(len(cFruit)) #输出：0，len()函数使用字典。
第12行
第13行 eFruit=("樱桃","椰子","石榴")
第14行 fFruit=dict.fromkeys(eFruit) #创建一个新字典。
第15行 print(fFruit) #输出：{'樱桃': None, '椰子': None, '石榴': None}
第16行
第17行 #如果Lechee已经存在于aFruit则返回字典中的值，否则返回第2个参数的值
第18行 print(aFruit.get("Lichee","荔枝")) #返回指定键的值
第19行 print(aFruit) #aFruit并没有发生改变
第20行
第21行 aFruit.setdefault("Lichee","荔枝") #如果不存在则添加该键，值为第2个参数
第22行 print(aFruit) #aFruit添加了"Lichee":"荔枝"
第23行
第24行 aFruit.setdefault("Lichee","荔枝A") #由于已存在，不添加
第25行 print(aFruit) #aFruit添加了"Lichee":"荔枝"
第26行
第27行 #用bFruit更新aFruit，如果aFruit不存在bFruit成员则增加，否则用bFruit更新aFruit
第28行 aFruit.update(bFruit)
第29行 print(aFruit) #Apple的值已经被更新为"平安果"
第30行
第31行 #删除键Grape对应的元素，如果不存在返回值为第2个参数
第32行 #存在则返回key对应的值
第33行 print(aFruit.pop("Grape","菩提子"))
第34行
第35行 aFruit.popitem() #随机删除一个成员
第36行 print(aFruit)
```

第二节　集合结构

集合set和字典dict很像，都是用花括号{}界定数据，都有键且键唯一，但集合有键无值，例程3-8是其基本示例。观察例程第3行会发现set也采用花括号{}界定数据数据成员，但相比dict，则每个成员不在是键值对，另外，从第4行的输出可以看出，虽然有两个"Apple"但已自动去重，集合的成员一定唯一。

观察第8行，要特别注意，空的集合必须如此写，如果写成cSet={}则申明cSet为字典而不是集合，而写成cSet=set()则是申明为空集合。

向集合增加成员用add()函数如例程第11行所示，删除集合中的成员用remove()函数，如例程第13行所示。

Python集合提供了集合运算，其定义与数学定义一直。&用于交集计算如例程第16行所示，计算出两个集合的共同部分成员；|用于并集计算如例程第17行所示，将两个集合合并去重后的所有成员；-用于差集计算如例程第18行所示，相当于从-符号左边

减去右边的成员后的左边剩余部分构成的集合，或者说减去两个成员的交集；^用于计算出两个差集，相当于集合a与集合b的差集与集合b与集合a差集的并集。

例程3-8

```
#结合set初步，集合成员不能重复，一定唯一

aSet={"Apple","Apple","Orange","Grape"}
print("输出：aSet:",aSet) #输出：{'Apple', 'Orange', 'Grape'}已自动去重

bSet={"Apple","Orange","Cherry"}

cSet=set() #空set，必须如此，不能cSet={}，这是空字典
print(cSet)

aSet.add("Lichee") #增加新成员
print("增加Lichee后，输出aSet:",aSet)
aSet.remove("Lichee") #删除已有成员
print("删除Lichee后，输出aSet:",aSet)

print(aSet&bSet) #&:交集，输出：{'Orange','Apple'}
print(aSet|bSet) #|:并集，输出：{'Orange','Apple','Grape','Cherry'}
print(aSet-bSet) #-:差集，输出：{'Grape'}，属于aSet但不属于bSet
print(aSet^bSet) #^:对称差集，输出：{'Grape', 'Cherry'}
```

```
输出: aSet: {'Grape', 'Apple', 'Orange'}
set()
增加Lichee后，输出aSet: {'Grape', 'Apple', 'Orange', 'Lichee'}
删除Lichee后，输出aSet: {'Grape', 'Apple', 'Orange'}
{'Apple', 'Orange'}
{'Grape', 'Apple', 'Cherry', 'Orange'}
{'Grape'}
{'Grape', 'Cherry'}
```

图3-5　例程3-8的执行效果

例程3-9是集合set的应用实例。第12行代码获得《三国演义》和《水浒传》的字数，用len()函数求得。第15行分别求得两本小说的字集（用了哪些字），并在第16行用并集求得两本小说的总字符集。从图3-6可以看出，两本小说的用字均不超过5000字，其中《三国演义》用字更多，达到4700余字。第21、22行用差集分别求出两本小说的没有用字的数量。第24-29行用字典分别记录下每个汉字的出现次数（字频），第32、33行则将每个汉字在两本小说中分别出现次数构成一个tuple成为对应的值。第35行将字典转换为list，第38行则按字频的差值排序，然后分别输出前10个(《三国演义》出现次数远高于《水浒传》)和后10个(与之相反)，细细观察，还是颇有意思。另外，处理时，没有清除标点符号等，会带来干扰，可以用代码清除之。

例程3-9

```
#计算《三国演义》和《水浒传》，集合应用示例
#sg开头表示《三国演义》，sh开头表示《水浒传》

SanGuoFile=open("三国演义.txt","r",encoding="utf-8")
sgData=SanGuoFile.read() #sgData存《三国演义》字符串
SanGuoFile.close()
```

```
第7行
第8行  ShuiHuFile=open("水浒传.txt","r",encoding="utf-8")
第9行  shData=ShuiHuFile.read() #shData存《水浒传》字符串
第10行 ShuiHuFile.close()
第11行
第12行 print(f"三国演义字数：{len(sgData)}，水浒传的字数：{len(shData)}")
第13行
第14行 #将字符串转换为集合，自动去重，相当于用了哪些字
第15行 sgHZ,shHZ=set(sgData),set(shData)
第16行 allHZ=sgHZ|shHZ #两本书全部用字集合，并集计算
第17行
第18行 print(f"三国演义字集：{len(sgHZ)}字,水浒传的字集：{len(shHZ)}字")
第19行 print(f"三国与水浒的共同字集：{len(allHZ)}")
第20行
第21行 print(f"三国没有用的字：{len(allHZ-sgHZ)}")
第22行 print(f"水浒没有用的字：{len(allHZ-shHZ)}")
第23行
第24行 sgDict,shDict,allDict={},{},{} #建立每个字的字典
第25行 for hz in sgData:
第26行     sgDict[hz]=sgDict.get(hz,0)+1 #建立《三国演义》每个字出现次数
第27行
第28行 for hz in shData:
第29行     shDict[hz]=shDict.get(hz,0)+1 #建立《水浒传》每个字出现次数
第30行
第31行 #建立每个字在三国演义和水浒传中出现次数，value为tuple
第32行 for hz in allHZ:
第33行     allDict[hz]=(sgDict.get(hz,0),shDict.get(hz,0))
第34行
第35行 allList=list(allDict.items())
第36行
第37行 #按字频相差排序
第38行 allList.sort(key=lambda x:x[1][0]-x[1][1],reverse=True)
第39行
第40行 print(allList[:10]) #前面10个，相当于在三国字频高，水浒字频低
第41行 print(allList[-10:]) #后面10个，与上面相反
第42行
第43行 #没有清除标点符号等，清除后效果更好
第44行 #eof
```

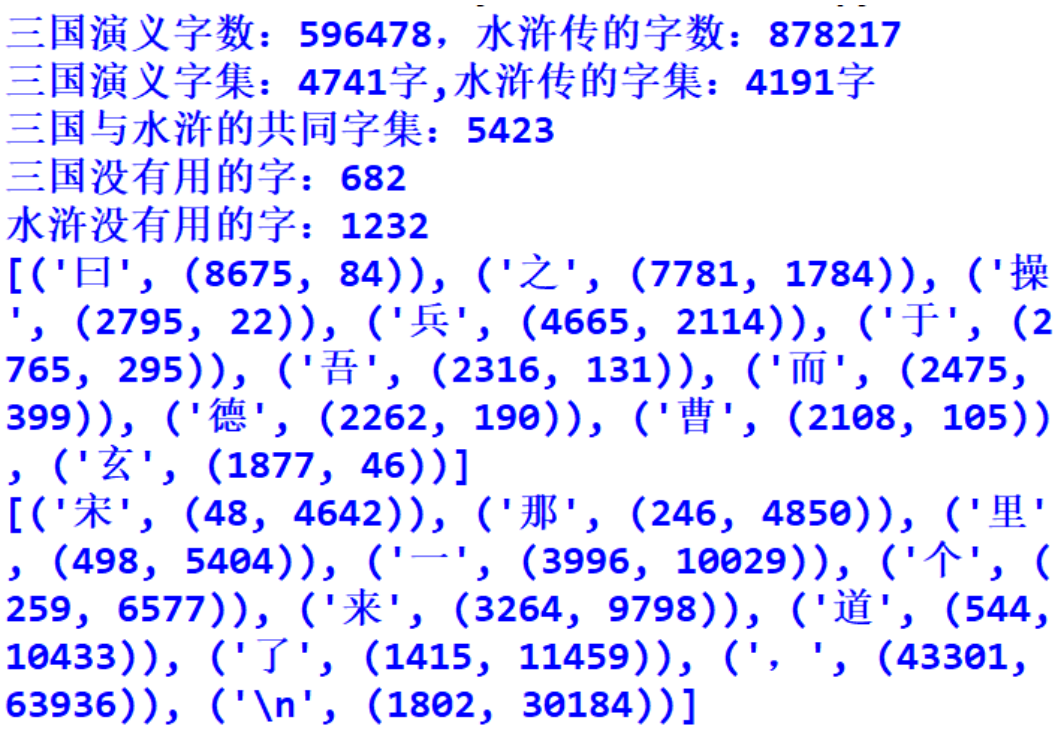

图3-6　例程3-9的执行效果

例程3-10是set类的其他一些常用函数。

例程3-10

```
第1行  #set的其他常用函数
第2行
第3行  aSet={"Apple","Apple","Orange","Grape"}
第4行  bSet={"Apple","Orange","Cherry"}
第5行  cSet={"Apple","Grape"}
第6行
第7行  print(cSet.issubset(aSet)) #输出：Ture，判断是否为子集
第8行  print(cSet.issuperset(aSet)) #输出：False，判断是否为父集
第9行
第10行 cSet.discard("Grape") #与remove()相似，都是删除一个成员。
第11行 #如果删除不存在成员，remove()发生异常，discard()不会。
第12行
第13行 print(aSet.isdisjoint(bSet)) #输出:False
第14行 #判断两个集合是否为分离集，不相交为分离集，返回值为True，相交False。
第15行 print(bSet.intersection(cSet)) #交集相当于&,输出：{'Apple'}
第16行 #difference=-、union=|、symmetric_difference=^，用法与之相同
第17行
第18行 bSet.update({"banana","Peach"}) #用集合更新集合，此处更新bSet
第19行 print(bSet)#输出：{'Orange','banana','Peach','Apple','Cherry'}
第20行
第21行 bSet.pop() #随机删除成员
第22行 bSet.clear() #清空所有成员，变成空集合
第23行
第24行 #eof
```

第三节　异同比较

list、tuple、set和dict是Python最常用的组合数据类型，都能一名多量存储大量数据，如何选择合适的数据类型呢？在例程3-11比较了list和dict的追加数据、查找数据和插入数据，其执行结果如图3-7所示，从图中可以看出向list或dict追加数据，效率相差不大，但是查找效率差别巨大。在dict查找数据，几乎不需要时间，而在list中查找数据，在耗时非常严重。向list插入1000个数据，与追加1000000个数据相比耗时也不少，但对于dict来讲，耗时几乎可以忽略不计。

相较于list，tuple不能增加删除修改数据，其查找机制与list相同，效率也不高。set和dict内部机制比较一致，其各方面执行效率也大致相同。在选择数据类型时，如果仅用于顺序遍历逐一访问处理，可以用list或tuple；如果有频繁的随机查找等，建议选择set或dict，但dict和set都不能存储相同的数据，要确保键的唯一。

例程3-11

```
第1行  #list和dict效率比较
第2行  import time
第3行  import random
第4行
第5行  dataList=[]
第6行  dataDict={}
第7行
第8行  #追加100,0000个随机数据，观察时间效率
第9行  startTime=time.process_time()
第10行
第11行 for i in range(0,1000000):
第12行     dataList.append((i,random.randint(0,10000)))
第13行
第14行 finishTime=time.process_time()
第15行 print("向list追加100,0000个数据耗时:",finishTime-startTime)
第16行
第17行 startTime=time.process_time()
第18行
第19行 for i in range(0,1000000):
第20行     dataDict[i]=random.randint(0,10000)
第21行
第22行 finishTime=time.process_time()
第23行 print("向dict追加100,0000个数据耗时:",finishTime-startTime)
第24行
第25行
第26行 findNumList=[] #用于保存1000个100到20000之间的随机整数，用于判断存在
第27行 for i in range(0,1000):
第28行     findNumList.append(random.randint(100,20000))
第29行
第30行 inState=True
第31行 startTime=time.process_time() #开始时间
第32行 for i in findNumList:
第33行     inState=i in dataList
第34行 finishTime=time.process_time() #结束时间
第35行 print("在list查找1000个数据耗时:",finishTime-startTime)
第36行
```

```
第37行  inState=True
第38行  startTime=time.process_time() #开始时间
第39行  for i in findNumList:
第40行      inState=i in dataDict
第41行  finishTime=time.process_time() #结束时间
第42行  print("在dict查找1000个数据耗时:",finishTime-startTime)
第43行
第44行  insertNumList=[] #用于保存1000个100到20000之间的随机整数，用于插入
第45行  for i in range(0,1000):
第46行      insertNumList.append(random.randint(100,20000))
第47行
第48行  startTime=time.process_time() #开始时间
第49行  for i in insertNumList:
第50行      dataList.insert(100,i)
第51行  finishTime=time.process_time() #结束时间
第52行  print("在list插入1000个数据耗时:",finishTime-startTime)
第53行
第54行  inState=True
第55行  startTime=time.process_time() #开始时间
第56行  for i in findNumList:
第57行      dataDict[i]=dataDict.get(i,0)+i
第58行  finishTime=time.process_time() #结束时间
第59行  print("在dict插入1000个数据耗时:",finishTime-startTime)
```

```
向list追加100,0000个数据耗时： 0.96875
向dict追加100,0000个数据耗时： 0.953125
在list查找1000个数据耗时： 15.25
在dict查找1000个数据耗时： 0.0
在list插入1000个数据耗时： 0.640625
在dict插入1000个数据耗时： 0.0
```

图3-7　例程3-11执行结果

表3-1是四种数据类型的综合比较，在使用过程中，可以根据需求选择不同的数据类型。

表3-1：list、tuple、set和dict的异同比较

功能	list	tuple	set	dict
括号类型	[]方括号	() 圆括号	{} 花括号仅有键	{} 花括号键值成对
是否有序	有序	有序	无序	无序
下标运算	支持	支持	不支持	支持，但不是编号而是key
成员数量	len()	len()	len()	len()
增加成员	append()	×	add() update() 【集合更新集合】	dictName[key]=value update() 【字典更新字典】
删除成员	remove() 只删除第一个匹配项	×	remove() discard()	pop()
更新成员值	listName[成员编号]=成员新值	×	×	dictName[key]=value
清空	clear()	×	clear()	clear()
存在判断 in、not in	支持	支持	支持	支持

切片支持 **其他**	**支持** **count(x):返回x出现次数** **index(x[, start[, end]]):返回指定值x出现位置** **insert(index,value):在指定位置index插入value值** **reverse():对list反序** **sort():排序，极常用。**	**支持** **支持**	**×** **特别关注：集合运算** **交集：intersection=&** **差集：difference=-** **并集：union=|** **对称差集：symmetric_difference=^**	**×** **特别关注：** **全部键：dictName.keys()** **全部值：dictName.values()** **全部条目：dictName.items()** **dictName.get(key,default)：获得key对应值，如果不存在，默认该值为default设定值。**

第四节　列表解析

根据已有支持for-in循环的数据创建新的组合数据类型，如例程3-12所示，第3行代码squareData被赋值为list类型，但和以往不同，并不是值的罗列，而是方括号内处理结果所构成的list，其内的for-in循环的每一个值i交给for-in前的表达式处理，其处理值形成新的list。在例程中，是将1-10逐一放到i中，然后交由i**2表达式处理，每个处理形成的值构成list的成员，所以第4行输出为结果为1-10的平方。类似形式即为列表解析。

列表解析支持if子句，其后表达式为真时的成员值将传递给for-in间的变量，对于例程而言，当i%5!=0的成员才会传递给i值，也才会被i**2处理形成list成员。if子句相当于对成员按条件过滤（filter）。

例程3-12

```
第1行  #列表解析
第2行
第3行  squareData=[i**2 for i in range(1,11)]
第4行  print(squareData)#输出：[1,4,9,16,25,36,49,64,81,100]
第5行
第6行  squareData=[i**2 for i in range(1,11) if i%5!=0]
第7行  print(squareData)#输出：[1,4,9,16,36,49,64,81]
第8行
第9行  #eof
```

例程3-13是对例程3-12第6-7行代码的传统实现，在列表解析中两行代码，在传统方式下需要5行代码。如果仅仅是输出1-10不能被5整除数的平方，一行代码即可实现：**print([i**2 for i in range(1,11) if i%5!=0])**，从中可以看出列表解析让代码更加简洁。

例程3-13

```
第1行  #列表解析
第2行  squareData=[]
第3行  for i in range(1,11):
第4行      if i%5!=0:
第5行          squareData.append(i**2)
第6行  print(squareData)
第7行  #eof
```

列表解析的其语法如下，其中iterData为支持for-in循环的数据，如：list、tuple、str、set、dict等，括号对可以是方括号list也可以是花括号set或dict，如例程3-14所示。var与其他for-in循环相同，expr通常是含有var的表达式，虽然可以没有。对于列表解析，if条件子句为可选项，可以有1个或者多个，也可以一个也没有。当存在时相当于对iterData过滤，但该if子句不能再有else子句。

```
[|{expr for var in iterData [if conditionExpr]}|]
#iterData为支持for-in循环的数据，方括号可以是其他括号,if子句为可选项。
```

例程3-14是列表解析的多种形式，要特别注意例程第19行tuple列表的形成，不能省去tuple，如果缺失，不会直接形成tuple型数据。

例程3-14

```
#列表解析
def isPrime(N): #判断一个数是否质数
    if N<2:return False
    for i in range(2,N):
        if N%i==0:return False
        if i*i>N:break
    return True

primeList=[i for i in range(1,30) if isPrime(i)==True] #形成list类型
print(primeList)#输出：[2,3,5,7,11,13,17,19,23,29]

primeSet={i for i in range(1,30) if isPrime(i)==True} #形成集合
print(primeSet)#输出：{2,3,5,7,11,13,17,19,23,29}

primeSquareDict={i:i**2 for i in range(1,30) if isPrime(i)==True} #形成dict
print(primeSquareDict)
{2:4,3:9,5:25,7:49,11:121,13:169,17:289,19:361,23:529,29:841}

primeTuple=tuple(i for i in range(1,30) if isPrime(i)==True) #形成tuple，特别注意，前面有tuple
print(primeTuple) #输出：(2,3,5,7,11,13,17,19,23,29)

#eof
```

例程3-15是列表解析的应用示例，用于过滤《红楼梦》文本中非汉字条目以及一些希望忽略的汉字(例程第15行ignoreHZ集合)。第16行代码：**if isChinese(hz) and hz not in ignoreHZ**用于判断变量hz是否为汉字以及是否在ignoreHZ集合中，如果为False则过滤。

例程3-15

```
#列表解析示例，《红楼梦》字频
def isChinese(ch):
    if 19968<=ord(ch)<40869:return True;
    return False
#unicode汉字编号范围：19968-40869，即十六进制范围4E00-9FA5。

hlmFile=open("红楼梦.txt","r",encoding="utf-8")
hlmText=hlmFile.read() #读取《红楼梦》全部文本
hlmFile.close()

hlmDict={}
for hz in hlmText:
    hlmDict[hz]=hlmDict.get(hz,0)+1 #计算每个字的出现次数

ignoreHZ=set("道了的不一来是人我你我他也着这有说去又们个在么得里只就上下")
hlmList=[(hz,rate)for (hz,rate) in hlmDict.items() if isChinese(hz) and hz not in ignoreHZ]
#由字典hlmDict形成list(列表)hlmList

```

```
    hlmList.sort(key=lambda x:x[1],reverse=True) #反序
第20行 print(hlmList[:30]) #仅列出前30
第21行
第22行 #eof
```

```
[('儿', 5928), ('玉', 5843), ('宝', 5635), ('子', 5301), ('贾', 5048),
('见', 4666), ('太', 4210), ('便', 3948), ('好', 3903), ('笑', 3802),
('家', 3800), ('姐', 3364), ('大', 3347), ('头', 3321), ('听', 3201),
('出', 3129), ('回', 3016), ('日', 2848), ('知', 2842), ('要', 2822),
('都', 2585), ('事', 2581), ('心', 2559), ('二', 2544), ('老', 2542),
('过', 2514), ('话', 2449), ('还', 2439), ('起', 2403), ('自', 2392)]
```

图3-8　例程3-15的执行效果

例程3-16是列表解析的更多示例，从代码第3行可以看出，列表解析可以有2个for-in，实际上可以多个，但超过3个一般不好良好的工程习惯。前置for-in成员将逐一与后随for-in根据表达式组合。每个for-in都可以有if子句（如例程第7行所示），可以跟随在for-in之后（如例程第7行所示），也可以集中写在最后（如例程第10行）。另外，for-in可以有多个if子句（如例程第13行所示），每个子句之间是and关系。

例程3-16

```
第1行 #列表解析
第2行
第3行 listA=[i+j for i in ("A","B","C","D") for j in ("X","Y","Z")]
第4行 print(listA)
第5行 #输出：['AX','AY','AZ','BX','BY','BZ','CX','CY','CZ','DX','DY','DZ']
第6行
第7行 listB=[i+j for i in range(1,11) if i%2==0 for j in range(1,11) if j%3==0]
第8行 print(listB) #输出：[5,8,11,7,10,13,9,12,15,11,14,17,13,16,19]
第9行
第10行 listC=[i+j for i in range(1,11) for j in range(1,11) if j%3==0 if i%2==0]
第11行 print(listC) #输出：[5,8,11,7,10,13,9,12,15,11,14,17,13,16,19]
第12行
第13行 listC=[i+j for i in range(1,11) for j in range(1,11) if j%3==0 if i%2==0 if j%2==0]
第14行 print(listC) #输出：[8,10,12,14,16]
第15行
第16行 #eof
```

例程3-17是列表解析一个应用示例，输出九九乘法表，从图3-9可以看出，数据（或者说等式）已经符合要求，但格式还没有达到预期，改进如例程3-18第3行所示，f-string内应用了条件表达式，根据条件输出不同格式，输出如图3-10所示。

例程3-17

```
第1行 #列表解析
第2行
第3行 tab99=[f"{j}*{i}={i*j}" for i in range(1,10) for j in range(1,i+1)]
第4行
第5行 for i in tab99:
第6行     print(i,end="")
第7行
第8行 #eof
```

```
1*1=11*2=22*2=41*3=32*3=63*3=91*4=42*4=83*4=124*4=161*5=52*5=103*
5=154*5=205*5=251*6=62*6=123*6=184*6=245*6=306*6=361*7=72*7=143*7
=214*7=285*7=356*7=427*7=491*8=82*8=163*8=244*8=325*8=406*8=487*8
=568*8=641*9=92*9=183*9=274*9=365*9=456*9=547*9=638*9=729*9=81
```

图3-9　例程3-17的执行效果

例程3-18

```
第1行  #列表解析
第2行
第3行  tab99=[f"{j}*{i}={i*j:<3}\n"if i==j else f"{j}*{i}={i*j:<3}" for i in range(1,10) for j in range(1,i+1)]
第4行
第5行  for i in tab99:
第6行      print(i,end="")
第7行
第8行  #eof
```

```
1*1=1
1*2=2  2*2=4
1*3=3  2*3=6  3*3=9
1*4=4  2*4=8  3*4=12 4*4=16
1*5=5  2*5=10 3*5=15 4*5=20 5*5=25
1*6=6  2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7  2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8  2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9  2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

图3-10　例程3-17的执行效果

第五节　类型转换

不同类型的数据之间可以互相转换，如已经学习过的str转换为int或float，以及int或float转换为str等。list、tuple、set以及dict之间也可以互相转换。另外，Python还有大量其他的数据类型可以转换。如例程3-19所示，在Python中，很多类型之间都可以互相转换，尤其是list、tuple、set和dict之间，为数据处理提供了极大方便，可以将这种转化理解为逐一取出每个成员，然后加上不同的括号。

例程3-19

```
第1行  #类型转换
第2行
第3行  numA=int("123")
第4行  strA=str(123)
第5行  numB=float("123.12")
第6行  numC=int(numB)
第7行  strB=str(123.12)
第8行  print(numA,strA,numB,numC,strB)
第9行
第10行 listA=list(range(1,100)) #将range()转换为list
第11行 tupleA=tuple(range(1,100)) #转换为tuple
第12行 setA=set(range(1,100)) #转换为set
第13行 dictA=dict([(1,1),(2,4),(3,9),(4,16)])
第14行 tupleB=tuple(dictA)
第15行 tupleC=tuple(dictA.items())
```

```
第16行  print(tupleB,tupleC,sep="###")#输出：(1,2,3,4)###((1,1),(2,4),(3,9),(4,16))
```

Python提供type()能获得直接量或间接量的类型，如例程3-20所示，从中可以看出，每个类型都是class（类），从侧面也证明Python是纯正面向对象程序设计。从第11行代码可以看出，dict.items()的类型是dict_items。从第13-14行代码可以看出，该类型支持for-in循环。从16行可以看出，该类型还可以转换为list，实际上也可以转换为其他类型。一般说来，只要支持for-in循环，都可以转换为tuple、list、set、dict等。

例程3-20

```
第1行  #类型转换
第2行
第3行  print(type("1234")) #输出：<class 'str'>
第4行  print(type(1234)) #输出：<class 'int'>
第5行  print(type(123.123)) #输出：<class 'float'>
第6行  print(type(False)) #输出：<class 'bool'>
第7行
第8行  print(type([1,2,3,4])) #输出：<class 'list'>
第9行  dictA={"Apple":"苹果","Cherry":"樱桃","Grape":"葡萄"}
第10行 print(type(dictA)) #输出：<class 'dict'>
第11行 print(type(dictA.items())) #输出：<class 'dict_items'>
第12行
第13行 for Every in dictA.items():
第14行     print(Every)
第15行
第16行 item2List=list(dictA.items())
第17行 print(item2List) #输出：[('Apple','苹果'),('Cherry','樱桃'),('Grape','葡萄')]
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'list'>
<class 'dict'>
<class 'dict_items'>
('Apple', '苹果')
('Cherry', '樱桃')
('Grape', '葡萄')
[('Apple', '苹果'), ('Cherry', '樱桃'), ('Grape', '葡萄')]
```

图3-11　例程3-20执行效果

支持for-in循环的数据集，还可以利用列表解析进行更加灵活的类型转换，如例程3-21所示，例程第4行将dict.items()用列表解析转换为list，例程第6行将range()产生的数经过过滤后转换为list，第7行还将其转换为str类型的list。例程第11-14行代码示例了如何将其他类型转换为字符串，以及字符串拼接等。

例程3-21

```
第1行  #类型转换
第2行
第3行  dictA={"Apple":"苹果","Cherry":"樱桃","Grape":"葡萄"}
第4行  listA=[(k,v) for (k,v) in dictA.items()]
第5行
第6行  listA=[i for i in range(1,100) if i%5==0]
第7行  listB=[str(i) for i in range(1,100) if i%5==0]
```

```
第8行  #listB的内容为range(1,100)每个成员转换为字符串后的list
第9行
第10行 #字符串函数可以将字符串成员的拼接成一个字符串。
第11行 newStr="->".join([str(i) for i in range(1,10)])
第12行 print(newStr) #输出：1->2->3->4->5->6->7->8->9
第13行 newStr="".join([str(i) for i in range(1,20)])
第14行 print(newStr) #输出：12345678910111213141516171819
```

第六节　数据函数

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3-22是其应用示例。

例程3-22

```
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3-23所示。

例程3-23

```
#filter()函数的使用
def isOdd(x):#判断是否奇数
    return x%2==1

listA=range(1,100)
listOdd=list(filter(isOdd,listA))#过滤掉偶数
#输出过滤后结果
print(listOdd)#输出：全部是奇数

listB=range(1,100)
listEven=list(filter(lambda x:x%2==0,listB))#过滤掉奇数
#输出过滤后结果
print(listEven)#输出：全部是偶数

#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3-24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。

例程3-24

```
#map()函数的使用

listB=range(1,10)
listMapA=list(map(lambda x:x+10,listB))#listB中每一个数都加上10
print(listMapA)

def numChange(N):#偶数乘以2，否则加上2
    if N%2==0:
        return N*2
    else:
        return N+2

listMapB=list(map(numChange,listB))#listB中每一个数都加上10
print(listMapB)

listA=[1,2,3,4]
listB=[5,6,7,8]
print(list(map(lambda x,y:x+y,listA,listB)))

#eof
```

map()函数支持多个序列，如例程3-24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3-25

```
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3-26所示。

例程3-26

```
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3-27所示。

例程3-27

```
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3-28所示。

例程3-28

```
#reversed()函数的使用示例
listA=[12,23,34,45,56]
reversedlistA=reversed(listA)
print(type(reversedlistA))#输出：<class 'list_reverseiterator'>
for i in reversedlistA:
    print(i,end="$$")#输出：56$$45$$34$$23$$12$$

#等价实现，会显得麻烦
for i in range(-1,-len(listA)-1,-1):
    print(listA[i],end="$$")#输出：56$$45$$34$$23$$12$$

strB="abcXYZ"
strC=reversed(strB)
for i in strC:
    print(i,end="$$")#输出：Z$$Y$$X$$c$$b$$a$$

#eof
```

第七节　题目示例

例程3-29

```
hlmFile=open(r"c:\PythonCode\hlm.txt","r",encoding="utf-8")
hlmData=hlmFile.read()
hlmFile.close() #关闭文件

hzRate={} #用于保存每个字符（主要是汉字）的出现频率（出现次数）,以每个汉字为key。
nGram=int(input("请输入nGram的n值"))
hzLast=hlmData[:nGram-1]
for cn in hlmData[nGram-1:]:
    hzNow=hzLast+cn
    if hzNow not in hzRate:
        hzRate[hzNow]=1
    else:
        hzRate[hzNow]+=1
    hzLast=hzLast[1:]+cn

#由于太多，直接print(hzRate)难以观察
hzRateList=list(hzRate.items())
hzRateList.sort(key=lambda x:x[1],reverse=True)
for (k,v) in hzRateList[:200]:
    for hz in k:
        if hz in "。，：　“”…！？\"":
            break
    else:
        print(f"{k}({v})",end=",")
```