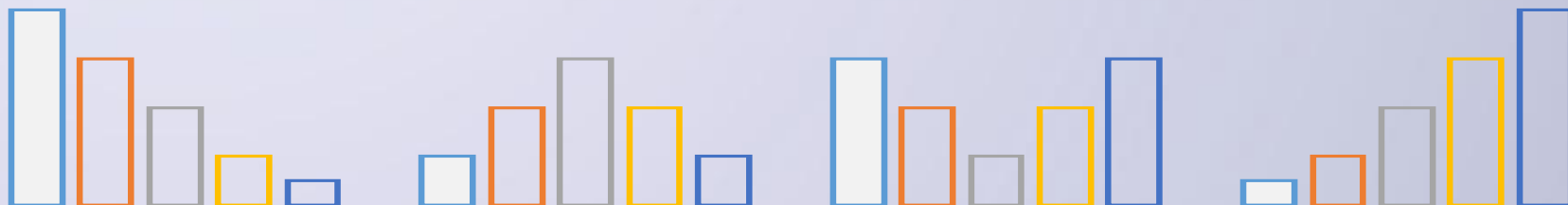
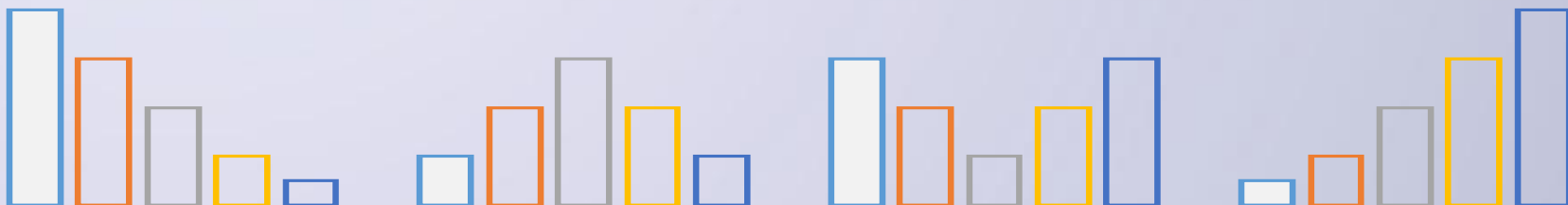


Python语言程序设计

北京理工大学 嵩天



第6章 组合数据类型





组合数据类型概述



序列类型

计算机不仅对单个变量表示的数据进行处理，更多情况，计算机需要对一组数据进行批量处理。一些例子包括：

- 给定一组单词{python, data, function, list, loop}，计算并输出每个单词的长度；
- 给定一个学院学生信息，统计一下男女生比例；
- 一次实验产生了很多组数据，对这些大量数据进行分析；



序列类型

组合数据类型能够将多个同类型或不同类型的数据组织起来，通过单一的表达使数据操作更有序更容易。根据数据之间的关系，组合数据类型可以分为三类：

序列类型、集合类型和映射类型。

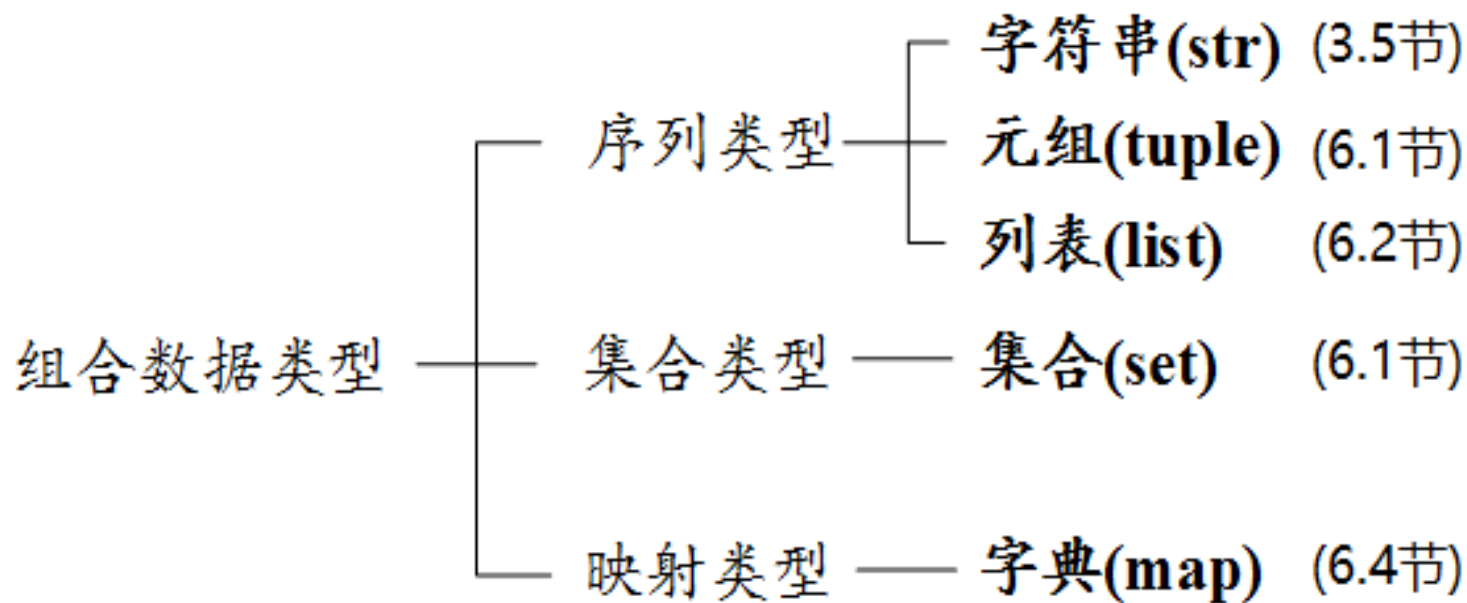


序列类型

- 序列类型是一个元素向量，元素之间存在先后关系，通过序号访问，元素之间不排他。
- 集合类型是一个元素集合，元素之间无序，相同元素在集合中唯一存在。
- 映射类型是“键-值”数据项的组合，每个元素是一个键值对，表示为(key, value)。



序列类型





序列类型

序列类型是一维元素向量，元素之间存在先后关系，通过序号访问。

当需要访问序列中某特定值时，只需要通过下标标出即可。

$$\sum_{i=0}^{n-1} S_i$$



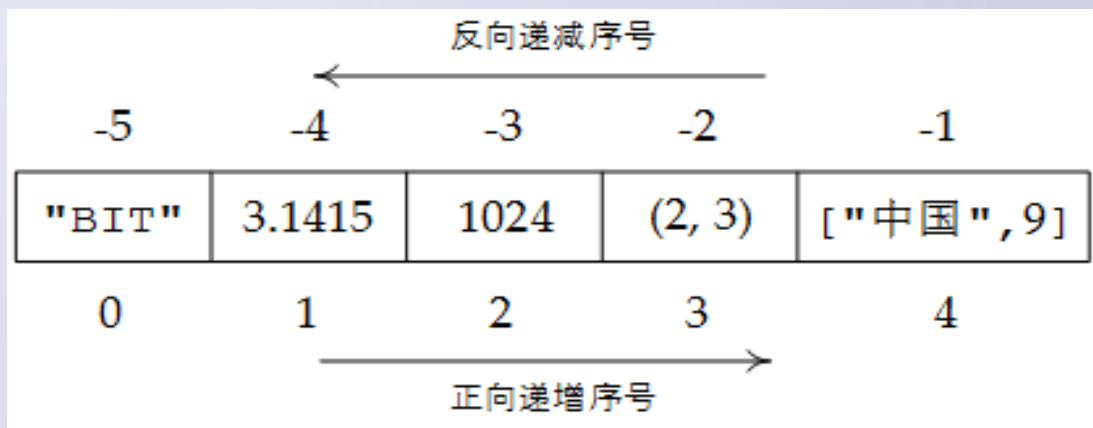
序列类型

由于元素之间存在顺序关系，所以序列中可以存在相同数值但位置不同的元素。序列类型支持成员关系操作符（`in`）、长度计算函数（`len()`）、分片（`[]`），元素本身也可以是序列类型。

序列类型

Python语言中有很多数据类型都是序列类型，其中比较重要的是：str（字符串）、tuple（元组）和list（列表）。

- 元组是包含0个或多个数据项的不可变序列类型。元组生成后是固定的，其中任何数据项不能替换或删除。
- 列表则是一个可以修改数据项的序列类型，使用也最灵活





序列类型

序列类型有12个通用的操作符和函数

操作符	描述
<code>x in s</code>	如果x是s的元素，返回True，否则返回False
<code>x not in s</code>	如果x不是s的元素，返回True，否则返回False
<code>s + t</code>	连接s和t
<code>s * n</code> 或 <code>n * s</code>	将序列s复制n次
<code>s[i]</code>	索引，返回序列的第i个元素
<code>s[i: j]</code>	分片，返回包含序列s第i到j个元素的子序列（不包含第j个元素）
<code>s[i: j: k]</code>	步骤分片，返回包含序列s第i到j个元素以j为步数的子序列
<code>len(s)</code>	序列s的元素个数（长度）
<code>min(s)</code>	序列s中的最小元素
<code>max(s)</code>	序列s中的最大元素
<code>s.index(x[, i[, j]])</code>	序列s中从i开始到j位置中第一次出现元素x的位置
<code>s.count(x)</code>	序列s中出现x的总次数



序列类型

元组 (tuple) 是序列类型中比较特殊的类型，因为它一旦创建就不能被修改。元组类型在表达固定数据项、函数多返回值、多变量同步赋值、循环遍历等情况下十分有用。Python中元组采用逗号和圆括号 (可选) 来表示。

```
>>>creature = "cat", "dog", "tiger", "human"
>>>creature
('cat', 'dog', 'tiger', 'human')
>>>color = ("red", 0x001100, "blue", creature)
>>>color
('red', 4352, 'blue', ('cat', 'dog', 'tiger', 'human'))
>>>color[2]
'blue'
>>>color[-1][2]
'tiger'
```



序列类型

```
>>>def func(x):  #函数多返回值
    return x, x**3

>>>a, b = 'dog', 'tiger'  #多变量同步赋值

>>>a, b = (b, a)          #多变量同步赋值，括号可省略

>>>import math

>>>for x, y in ((1,0), (2,5), (3,8)):  #循环遍历
    print(math.hypot(x,y))            #求多个坐标值到原点的距离
```



集合类型

集合类型与数学中集合的概念一致，即包含0个或多个数据项的无序组合。集合中元素不可重复，元素类型只能是固定数据类型，例如：整数、浮点数、字符串、元组等，列表、字典和集合类型本身都是可变数据类型，不能作为集合的元素出现。



集合类型

由于集合是无序组合，它没有索引和位置的概念，不能分片，集合中元素可以动态增加或删除。集合用大括号（{}）表示，可以用赋值语句生成一个集合。

```
>>>S = {425, "BIT", (10, "CS"), 424}
>>>S
{424, 425, (10, 'CS'), 'BIT'}
>>>T = {425, "BIT", (10, "CS"), 424, 425, "BIT"}
>>>T
{424, 425, (10, 'CS'), 'BIT'}
```



集合类型

由于集合元素是无序的，集合的打印效果与定义顺序可以不一致。由于集合元素独一无二，使用集合类型能够过滤掉重复元素。set(x)函数可以用于生成集合。

```
>>>w = set('apple')  
{ 'e', 'p', 'a', 'l' }  
  
>>>v = set(("cat", "dog", "tiger", "human"))  
{ 'cat', 'human', 'dog', 'tiger' }
```



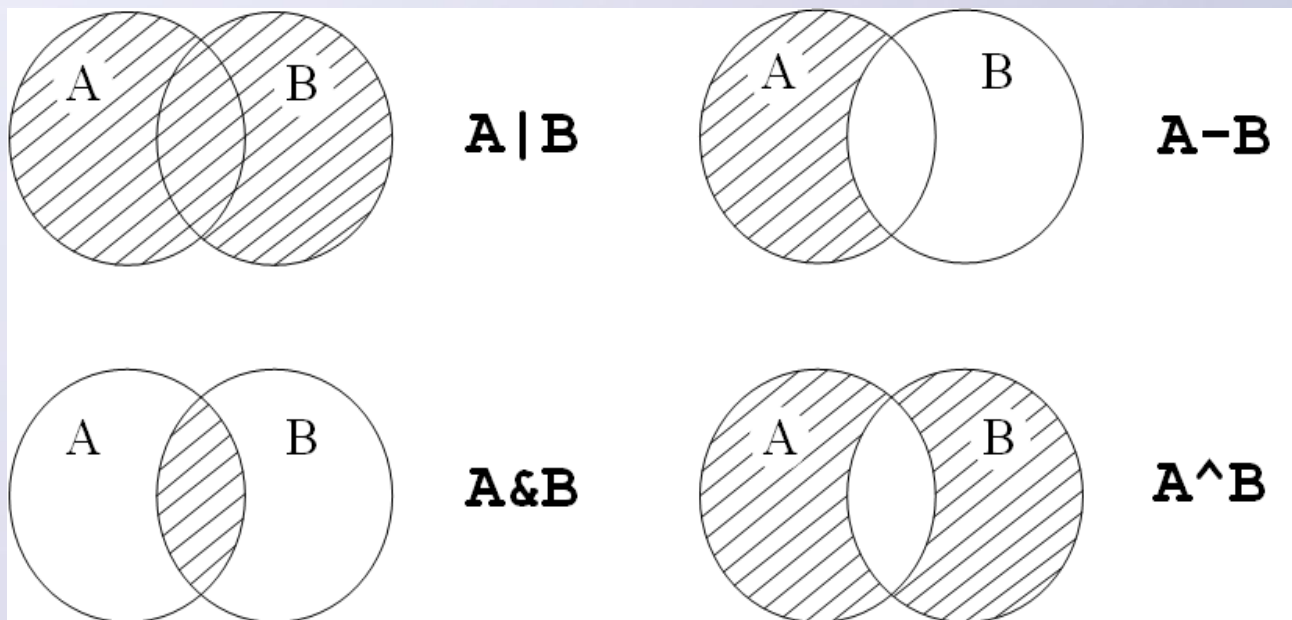

集合类型

集合类型有10个操作符

操作符	描述
$S - T$ 或 <code>S.difference(T)</code>	返回一个新集合，包括在集合S中但不在集合T中的元素
$S -= T$ 或 <code>S.difference_update(T)</code>	更新集合S，包括在集合S中但不在集合T中的元素
$S \& T$ 或 <code>S.intersection(T)</code>	返回一个新集合，包括同时在集合S和T中的元素
$S \&= T$ 或 <code>S.intersection_update(T)</code>	更新集合S，包括同时在集合S和T中的元素。
$S \wedge T$ 或 <code>s.symmetric_difference(T)</code>	返回一个新集合，包括集合S和T中元素，但不包括同时在其中的元素
$S \wedge= T$ 或 <code>s.symmetric_difference_update(T)</code>	更新集合S，包括集合S和T中元素，但不包括同时在其中的元素
$S T$ 或 <code>S.union(T)</code>	返回一个新集合，包括集合S和T中所有元素
$S = T$ 或 <code>S.update(T)</code>	更新集合S，包括集合S和T中所有元素
$S \leq T$ 或 <code>S.issubset(T)</code>	如果S与T相同或S是T的子集，返回True，否则返回False，可以用 $S < T$ 判断S是否是T的真子集
$S \geq T$ 或 <code>S.issuperset(T)</code>	如果S与T相同或S是T的超集，返回True，否则返回False，可以用 $S > T$ 判断S是否是T的真超集

集合类型

上述操作符表达了集合类型的4种基本操作，交集（&）、并集（|）、差集（-）、补集（^），操作逻辑与数学定义相同





集合类型

集合类型有10个操作函数或方法

函数或方法	描述
<code>S.add(x)</code>	如果数据项 <code>x</code> 不在集合 <code>S</code> 中，将 <code>x</code> 增加到 <code>s</code>
<code>S.clear()</code>	移除 <code>S</code> 中所有数据项
<code>S.copy()</code>	返回集合 <code>S</code> 的一个拷贝
<code>S.pop()</code>	随机返回集合 <code>S</code> 中的一个元素，如果 <code>S</code> 为空，产生 <code>KeyError</code> 异常
<code>S.discard(x)</code>	如果 <code>x</code> 在集合 <code>S</code> 中，移除该元素；如果 <code>x</code> 不在，不报错
<code>S.remove(x)</code>	如果 <code>x</code> 在集合 <code>S</code> 中，移除该元素；不在产生 <code>KeyError</code> 异常
<code>S.isdisjoint(T)</code>	如果集合 <code>S</code> 与 <code>T</code> 没有相同元素，返回 <code>True</code>
<code>len(S)</code>	返回集合 <code>S</code> 元素个数
<code>x in S</code>	如果 <code>x</code> 是 <code>S</code> 的元素，返回 <code>True</code> ，否则返回 <code>False</code>
<code>x not in S</code>	如果 <code>x</code> 不是 <code>S</code> 的元素，返回 <code>True</code> ，否则返回 <code>False</code>



集合类型

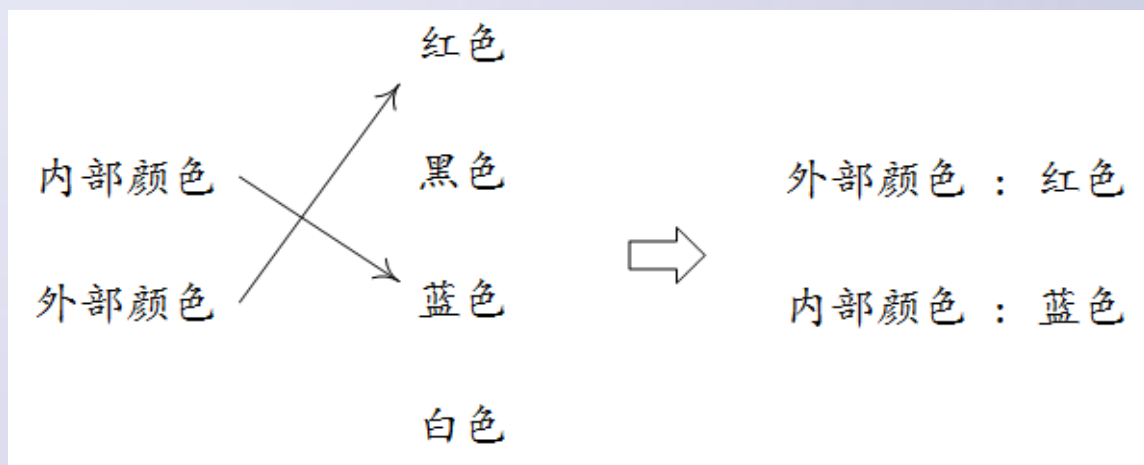
集合类型主要用于三个场景：成员关系测试、元素去重和删除数据项，例子如下。

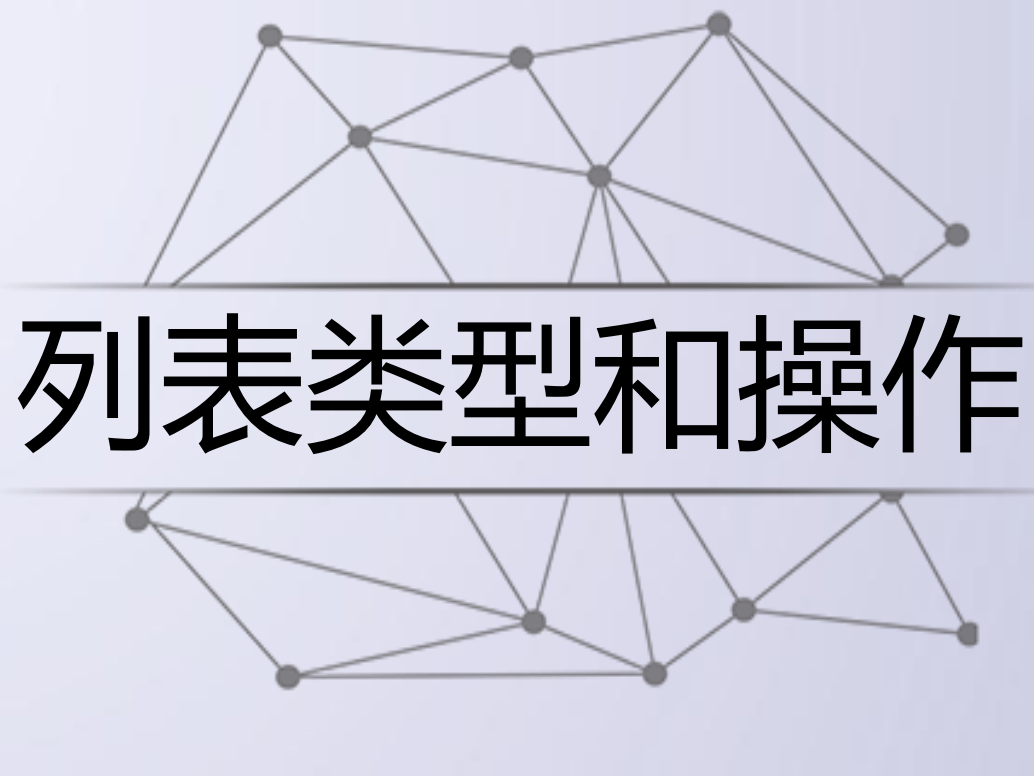
```
>>>"BIT" in {"PYTHON", "BIT", 123, "GOOD"} #成员关系测试
True
>>>tup = ("PYTHON", "BIT", 123, "GOOD", 123) #元素去重
>>>set(tup)
{123, 'GOOD', 'BIT', 'PYTHON'}
>>>newtup = tuple(set(tup)-{'PYTHON'}) # 去重同时删除数据项
('GOOD', 123, 'BIT')
```

集合类型与其他类型最大的不同在于它不包含重复元素，因此，当需要对一维数据进行去重或进行数据重复处理时，一般通过集合来完成。


映射类型

映射类型是“键-值”数据项的组合，每个元素是一个键值对，即元素是(key, value)，元素之间是无序的。键值对(key, value)是一种二元关系。在Python中，映射类型主要以字典（dict）体现。






列表类型和操作



列表类型的概念

列表（list）是包含0个或多个对象引用的有序序列，属于序列类型。与元组不同，列表的长度和内容都是可变的，可自由对列表中数据项进行增加、删除或替换。列表没有长度限制，元素类型可以不同，使用非常灵活。



列表类型的概念

由于列表属于序列类型，所以列表也支持成员关系操作符（`in`）、长度计算函数（`len()`）、分片（`[]`）。列表可以同时使用正向递增序号和反向递减序号，可以采用标准的比较操作符（`<`、`<=`、`==`、`!=`、`>=`、`>`）进行比较，列表的比较实际上是单个数据项的逐个比较。



列表类型的概念

列表用中括号（ [] ）表示，也可以通过list()函数将元组或字符串转化成列表。直接使用list()函数会返回一个空列表。

```
>>>ls = [425, "BIT", [10, "CS"], 425]
>>>ls
[425, 'BIT', [10, 'CS'], 425]
>>>ls[2][-1][0]
'C'
>>>list((425, "BIT", [10, "CS"], 425))
[425, 'BIT', [10, 'CS'], 425]
>>>list("中国是一个伟大的国家")
['中', '国', '是', '一', '个', '伟', '大', '的', '国', '家']
>>>list()
[]
```



列表类型的概念

与整数和字符串不同，列表要处理一组数据，因此，列表必须通过显式的数据赋值才能生成，简单将一个列表赋值给另一个列表不会生成新的列表对象。

```
>>>ls = [425, "BIT", 1024]  #用数据赋值产生列表ls
>>>lt = ls                  #lt是ls所对应数据的引用，lt并不包含真实数据
>>>ls[0] = 0
>>>lt
[0, 'BIT', 1024]
```

列表类型的概念

```
>>>ls = [425, "BIT", 1024]
```

```
>>>lt = ls
```

```
>>>ls[0] = 0
```

```
>>>lt
```




ls

lt

[425, "BIT", 1024]

0



列表类型的操作

函数或方法	描述
<code>ls[i] = x</code>	替换列表ls第i数据项为x
<code>ls[i: j] = lt</code>	用列表lt替换列表ls中第i到j项数据（不含第j项，下同）
<code>ls[i: j: k] = lt</code>	用列表lt替换列表ls中第i到j以k为步的数据
<code>del ls[i: j]</code>	删除列表ls第i到j项数据，等价于 <code>ls[i: j]=[]</code>
<code>del ls[i: j: k]</code>	删除列表ls第i到j以k为步的数据
<code>ls += lt</code> 或 <code>ls.extend(lt)</code>	将列表lt元素增加到列表ls中
<code>ls *= n</code>	更新列表ls，其元素重复n次
<code>ls.append(x)</code>	在列表ls最后增加一个元素x
<code>ls.clear()</code>	删除ls中所有元素
<code>ls.copy()</code>	生成一个新列表，复制ls中所有元素
<code>ls.insert(i, x)</code>	在列表ls第i位置增加元素x
<code>ls.pop(i)</code>	将列表ls中第i项元素取出并删除该元素
<code>ls.remove(x)</code>	将列表中出现的第一个元素x删除
<code>ls.reverse(x)</code>	列表ls中元素反转



列表类型的操作

```
>>>vlist = list(range(5))

>>>vlist

[0, 1, 2, 3, 4]

>>>len(vlist[2:])          #计算从第3个位置开始到结尾的子串长度

3

>>>2 in vlist              #判断2是否在列表vlist中

True

>>>vlist[3]="python"      #修改序号3的元素值和类型

>>>vlist

[0, 1, 2, 'python', 4]

>>>vlist[1:3]=["bit", "computer"]

>>>vlist

[0, 'bit', 'computer', 3, 4]
```



列表类型的操作


当使用一个列表改变另一个列表值时，Python不要求两个列表长度一样，但遵循“多增少减”的原则，例子如下。

```
>>>vlist[1:3]=["new_bit", "new_computer", 123]

>>>vlist
[0, 'new_bit', 'new_computer', 123, 'python', 4]

>>>vlist[1:3]=["fewer"]

>>>vlist
[0, 'fewer', 123, 'python', 4]
```



列表类型的操作

与元组一样，列表可以通过for...in语句对其元素进行遍历，基本语法结构如下：

```
for    <任意变量名> in <列表名>:  
    语句块
```



列表类型的操作

```
>>>for e in vlist:  
    print(e, end=" ")  
0 fewer 123 python 4
```

列表是一个十分灵活的数据结构，它具有处理任意长度、混合类型的能力，并提供了丰富的基础操作符和方法。当程序需要使用组合数据类型管理批量数据时，请尽量使用列表类型。



基本统计值计算



基本统计值的计算

以最简单的统计问题为例，求解一组不定长数据的基本统计值，即平均值、标准差、中位数。

一组数据表示为 $S = s_0, s_1, \dots, s_{n-1}$ ，其算术平均值、标准差分别表示为：

$$m = \left(\sum_{i=0}^{n-1} s_i \right) / n \quad \text{和} \quad d = \sqrt{\left(\sum_{i=0}^{n-1} (s_i - m)^2 \right) / (n - 1)}$$



基本统计值的计算

由于平均数、标准差和中位数是三个不同的计算目标，使用函数方式编写计算程序。

`getNum()`函数从用户输入获得数据

`mean()`函数计算平均值

`dev()`函数计算标准差

`median()`函数计算中位数



基本统计值的计算

实例代码9.1

e9.1CalStatistics.py

```
1  #e9.1CalStatistics.py
2  from math import sqrt
3  def getNum():          #获取用户输入
4      nums = []
5      iNumStr = input("请输入数字(直接输入回车退出): ")
6      while iNumStr != "":
7          nums.append(eval(iNumStr))
8          iNumStr = input("请输入数字(直接输入回车退出): ")
9      return nums
10
11 def mean(numbers):      #计算平均值
12     s = 0.0
13     for num in numbers:
14         s = s + num
15     return s / len(numbers)
```



基本统计值的计算

实例代码9.1

e9.1CalStatistics.py

```
15 def dev(numbers, mean): #计算方差
16     sdev = 0.0
17     for num in numbers:
18         sdev = sdev + (num - mean)**2
19     return sqrt(sdev / (len(numbers)-1))
20 def median(numbers):      #计算中位数
21     sorted(numbers)
22     size = len(numbers)
23     if size % 2 == 0:
24         med = (numbers[size//2-1] + numbers[size//2])/2
25     else:
26         med = numbers[size//2]
27     return med
28 n =  getNum()  #主体函数
29 m =  mean(n)
30 print("平均值:{},方差:{:.2},中位数:{}.".format(m,\
    dev(n,m),median(n)))
```



基本统计值的计算

```
>>>  
请输入数字(直接输入回车退出): 99  
请输入数字(直接输入回车退出): 98  
请输入数字(直接输入回车退出): 97  
请输入数字(直接输入回车退出): 96  
请输入数字(直接输入回车退出): 95  
请输入数字(直接输入回车退出):  
平均值:97.0, 方差:1.6, 中位数:97.
```


程序先后调用getNum()、mean()、dev()和median()函数。利用函数的模块化设计能够复用代码并增加代码的可读性。每个函数内部都采用了简单的语句。




基本统计值的计算

列表在实现基本数据统计时发挥了重要作用，表现在：

- 列表是一个动态长度的数据结构，可以根据需求增加或减少元素；
- 列表的一系列方法或操作符为计算提供了简单的元素运算手段；
- 列表提供了对每个元素的简单访问方式及所有元素的遍历方式。



字典类型的计算



字典类型的基本概念

通过任意键信息查找一组数据中值信息的过程叫映射，Python语言中通过字典实现映射。Python语言中的字典可以通过大括号({})建立，建立模式如下：

{<键1>:<值1>, <键2>:<值2>, ..., <键n>:<值n>}

其中，键和值通过冒号连接，不同键值对通过逗号隔开。



字典类型的基本概念

```
>>>Dcountry={"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}  
>>>print(Dcountry)  
{'中国': '北京', '法国': '巴黎', '美国': '华盛顿'}
```

字典打印出来的顺序与创建之初的顺序不同，这不是错误。字典是集合类型的延续，各个元素并没有顺序之分。



字典类型的基本概念

字典最主要的用法是查找与特定键相对应的值，这通过索引符号来实现。

```
>>>Dcountry["中国"]  
'北京'
```

一般来说，字典中键值对的访问模式如下，采用中括号格式：

<值> = <字典变量>[<键>]

字典中对某个键值的修改可以通过中括号的访问和赋值实现：

```
>>>Dcountry["中国"]='大北京'  
>>>print(Dcountry)  
{ '中国': '大北京', '法国': '巴黎', '美国': '华盛顿' }
```



字典类型的操作

通过中括号可以增加新的元素

```
>>>Dcountry={"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>>Dcountry["英国"]="伦敦"
>>>print(Dcountry)
{'中国': '北京', '法国': '巴黎', '美国': '华盛顿', '英国': '伦敦'}
```

直接使用大括号 ({}) 可以创建一个空的字典，并通过中括号 ([]) 向其增加元素。

```
>>>Dp={ }
>>>Dp['2^10']=1024
>>>print(Dp)
{'2^10': 1024}
```



字典类型的操作

函数和方法	描述
<code><d>.keys()</code>	返回所有的键信息
<code><d>.values()</code>	返回所有的值信息
<code><d>.items()</code>	返回所有的键值对
<code><d>.get(<key>,<default>)</code>	键存在则返回相应值，否则返回默认值
<code><d>.pop(<key>,<default>)</code>	键存在则返回相应值，同时删除键值对，否则返回默认值
<code><d>.popitem()</code>	随机从字典中取出一个键值对，以元组(key, value)形式返回
<code><d>.clear()</code>	删除所有的键值对
<code>del <d>[<key>]</code>	删除字典中某一个键值对
<code><key> in <d></code>	如果键在字典中返回True，否则返回False



字典类型的操作

```
>>>Dcountry={"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>>Dcountry.keys()
dict_keys(['中国', '美国', '法国'])
>>>list(Dcountry.values())
['北京', '华盛顿', '巴黎']
>>>Dcountry.items()
dict_items([('中国', '北京'), ('美国', '华盛顿'), ('法国', '巴黎')])
>>>'中国' in Dcountry    #只对键进行判断
True
>>>Dcountry.get('美国', '悉尼') #'美国'在字典中存在
'华盛顿'
>>>Dcountry.get('澳大利亚', '悉尼') #'澳大利亚'在字典中不存在
'悉尼'
```



字典类型的操作

与其他组合类型一样，字典可以通过for...in语句对其元素进行遍历，基本语法结构如下：

for <变量名> in <字典名>:
 语句块

```
>>>for key in Dcountry:  
    print(key)
```

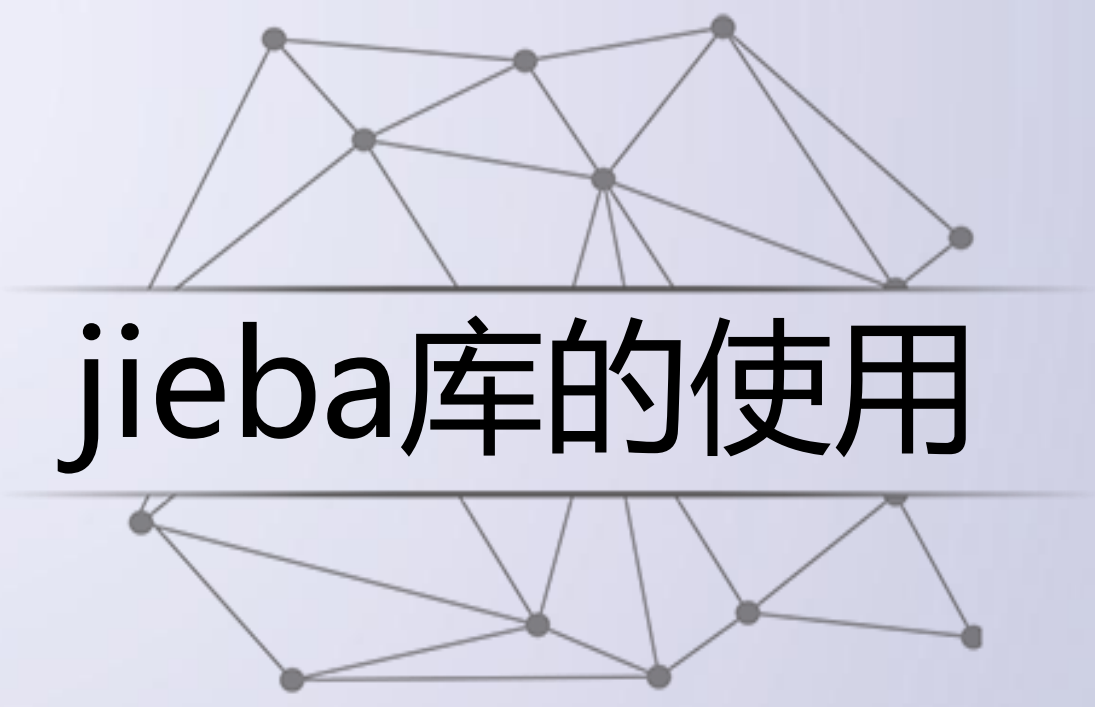
```
中国  
美国  
法国
```



字典类型的操作

字典是实现键值对映射的数据结构，请理解如下基本原则：

- 字典是一个键值对的集合，该集合以键为索引，一个键信息只对应一个值信息；
- 字典中元素以键信息为索引访问；
- 字典长度是可变的，可以通过对键信息赋值实现增加或修改键值对。



jieba库的使用



jieba库的概述

jieba是Python中一个重要的第三方中文分词函数库

```
>>>import jieba  
>>>jieba.lcut("中国是一个伟大的国家")  
['中国', '是', '一个', '伟大', '的', '国家']
```

jieba库是第三方库，不是安装包自带，需要通过pip指令安装

```
:>pip install jieba # 或者 pip3 install jieba
```



jieba库的解析

函数	描述
<code>jieba.cut(s)</code>	精确模式，返回一个可迭代的数据类型
<code>jieba.cut(s, cut_all=True)</code>	全模式，输出文本s中所有可能单词
<code>jieba.cut_for_search(s)</code>	搜索引擎模式，适合搜索引擎建立索引的分词结果
<code>jieba.lcut(s)</code>	精确模式，返回一个列表类型，建议使用
<code>jieba.lcut(s, cut_all=True)</code>	全模式，返回一个列表类型，建议使用
<code>jieba.lcut_for_search(s)</code>	搜索引擎模式，返回一个列表类型，建议使用
<code>jieba.add_word(w)</code>	向分词词典中增加新词w



jieba库的解析

```
>>>import jieba
```

```
>>>jieba.lcut("中华人民共和国是一个伟大的国家")
```

```
['中华人民共和国', '是', '一个', '伟大', '的', '国家']
```

```
>>>jieba.lcut("中华人民共和国是一个伟大的国家", cut_all=True)
```

```
['中华', '中华人民', '中华人民共和国', '华人', '人民', '人民共和国', '共和', '共和国', '国是', '一个', '伟大', '的', '国家']
```

```
>>>jieba.lcut_for_search("中华人民共和国是一个伟大的国家")
```

```
['中华', '华人', '人民', '共和', '共和国', '中华人民共和国', '是', '一个', '伟大', '的', '国家']
```



文本词频统计



《Hamlet》英文词频统计

实例代码10.1 e10.1CalHamlet.py

```
1  #e10.1CalHamlet.py
2  def getText():
3      txt = open("hamlet.txt", "r").read()
4      txt = txt.lower()
5      for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
6          txt = txt.replace(ch, " ")    #将文本中特殊字符替换为空格
7      return txt
8      hamletTxt = getText()
9      words  = hamletTxt.split()
10     counts = {}
11     for word in words:
12         counts[word] = counts.get(word,0) + 1
13     items = list(counts.items())
14     items.sort(key=lambda x:x[1], reverse=True)
15     for i in range(10):
16         word, count = items[i]
17         print ("{:<10}{1:>5}".format(word, count))
```



《Hamlet》英文词频统计

```
>>>
the          1138
and           965
to            754
of            669
you           550
a             542
i             542
my            514
hamlet        462
in            436
```

观察输出结果可以看到，高频单词大多数是冠词、代词、连接词等语法型词汇，并不能代表文章的含义。进一步，可以采用集合类型构建一个排除词汇库excludes，在输出结果中排除这个词汇库中内容。

```
1  #e10.2CalHamlet.py
2  excludes = {"the","and","of","you","a","i","my","in"}
3  def getText():
4      txt = open("hamlet.txt", "r").read()
5      txt = txt.lower()
6      for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
7          txt = txt.replace(ch, " ")    #将文本中特殊字符替换为空格
8      return txt
9  hamletTxt = getText()
10 words = hamletTxt.split()
11 counts = {}
12 for word in words:
13     counts[word] = counts.get(word,0) + 1
14 for word in excludes:
15     del(counts[word])
16 items = list(counts.items())
17 items.sort(key=lambda x:x[1], reverse=True)
18 for i in range(10):
19     word, count = items[i]
20     print ("{0:<10}{1:>5}".format(word, count))
```




《Hamlet》英文词频统计

运行程序后，输出结果如下

```
>>>  
to          754  
hamlet      462  
it          416  
that        391  
is          340  
not         314  
lord        309  
his         296  
this        295  
but         269
```



《三国演义》人物出场统计

实例代码10.3

e10.3CalThreeKingdoms.py

```
1  #e10.3CalThreeKingdoms.py
2  import jieba
3  txt = open("三国演义.txt", "r", encoding='utf-8').read()
4  words = jieba.lcut(txt)
5  counts = {}
6  for word in words:
7      if len(word) == 1: #排除单个字符的分词结果
8          continue
9      else:
10         counts[word] = counts.get(word,0) + 1
11 items = list(counts.items())
12 items.sort(key=lambda x:x[1], reverse=True)
13 for i in range(15):
14     word, count = items[i]
15 print ("{0:<10}{1:>5}".format(word, count))
```



《三国演义》人物出场统计

>>>

曹操	953
孔明	836
将军	772
却说	656
玄德	585
关公	510
丞相	491
二人	469
不可	440
荆州	425
玄德曰	390
孔明曰	390
不能	384
如此	378
张飞	358



《三国演义》人物出场统计

观察输出结果，同一个人物会有不同的名字，这种情况需要整合处理。同时，与英文词频统计类似，需要排除一些人名无关词汇，如“却说”、“将军”等。



《三国演义》人物出场统计

实例代码10.4 e10.4CalThreeKingdoms.py

```
1 #e10.4CalThreeKingdoms.py
2 import jieba
3 excludes = {"将军","却说","荆州","二人","不可","不能","如此"}
4 txt = open("三国演义.txt", "r", encoding='utf-8').read()
5 words = jieba.lcut(txt)
6 counts = {}
7 for word in words:
8     if len(word) == 1:
9         continue
10    elif word == "诸葛亮" or word == "孔明曰":
11        rword = "孔明"
12    elif word == "关公" or word == "云长":
13        rword = "关羽"
14    elif word == "玄德" or word == "玄德曰":
15        rword = "刘备"
16    elif word == "孟德" or word == "丞相":
17        rword = "曹操"
```



《三国演义》人物出场统计

实例代码10.4 e10.4CalThreeKingdoms.py

```
18 else:
19     rword = word
20     counts[rword] = counts.get(rword,0) + 1
21 for word in excludes:
22     del(counts[word])
23 items = list(counts.items())
24 items.sort(key=lambda x:x[1], reverse=True)
25 for i in range(5):
26     word, count = items[i]
27     print ("{:<10}{1:>5}".format(word, count))
```



《三国演义》人物出场统计

输出排序前5的单词，运行程序后，输出结果如下：

```
>>>
曹操          1451
孔明          1383
刘备          1252
关羽           784
张飞           358
```

请继续完善程序，排除更多无关词汇干扰，总结出场最多的20个人物都有哪些。这里，给出参考答案。

曹操（1451）、孔明（1383）、刘备（1252）、关羽（784）、张飞（358）、
吕布（300）、赵云（278）、孙权（264）、司马懿（221）、周瑜（217）、
袁绍（191）、马超（185）、魏延（180）、黄忠（168）、姜维（151）、
马岱（127）、庞德（122）、孟获（122）、刘表（120）、夏侯惇（116）



Python之禅



Python之禅

什么样的程序是好的？如何编写漂亮的代码？这都是学习编程一段时间最经常提出的问题，却最难回答。程序设计语言如同自然语言，好的代码就像文学作品，不仅达意，更要优美。那什么是“好”？什么是“优美”？领悟编程代码优美的过程类似参禅，除了不断练习，也需要理解一些原则。



Python之禅

Python编译器以函数库的形式内置了一个有趣的文件，被称为“Python之禅”（The Zen of Python）。当调用如下一行语句后，会出现一段有趣的运行结果。

```
>>>import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!



Python之禅

作者: Tim Peters

- 优美胜于丑陋
- 明了胜于隐晦
- 简洁胜于复杂
- 复杂胜于凌乱
- 扁平胜于嵌套
- 间隔胜于紧凑
- 可读性很重要
- 即便假借特例的实用性之名，也不要违背上述规则
- 除非你确定需要，任何错误都应该有应对
- 当存在多种可能，不要尝试去猜测
- 只要你不是Guido，对于问题尽量找一种，最好是唯一明显的解决方案
- 做也许好过不做，但不假思索就动手还不如不做
- 如果你无法向人描述你的实现方案，那肯定不是一个好方案
- 如果实现方案容易解释，可能是个好方案
- 命名空间是绝妙的理念，要多运用

译者心得

以编写优美代码为目标，不多解释
优美代码应该清晰明了，规范统一
优美代码应该逻辑简洁，避免复杂逻辑
如果必须采用复杂逻辑，接口关系也要清晰
优美代码应该是扁平的，避免太多层次嵌套
优美代码间隔要适当，每行代码解决适度问题
优美代码必须是可读且易读的
上述规则是至高无上的

捕获异常，不让程序留有因错误退出的可能

不要试图给出多种方案，找到一种实现它，几乎所有人没有Guido那么牛
编程之前要有思考

能说清楚的往往才是对的

适合复杂程序编程



Python之禅

除了Python之禅所表达的Python设计理念，该程序还有另一段魅力：

实例代码11.1

this.py

```
1 s = """Gur Mra bs Clguba, ol Gvz Crgref
2
3 Ornhgvshy vf orggre guna htyl.
4 Rkcyvpgv vf orggre guna vzcypvg.
5 Fvzcyr vf orggre guna pbzcyrk.
6 Pbzcyrk vf orggre guna pbzcyvpngrq.
7 Syng vf orggre guna arfgrq.
8 Fcnefr vf orggre guna qrafr.
9 Ernqnovyvgl pbhagf.
10 Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehurf.
11 Nygubhtu cenpgvpnyvgl orngf chevgl.
12 Reebef fubhyq arire cnff fvyragyl.
13 Hayrff rkcyvpvgyl fvyraprq.
```



Python之禅

实例代码11.1

this.py

```
14 Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
15 Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
16 Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
17 Abj vf orggre guna arire.
18 Nygubhtu arire vf bsgra orggre guna *evtug* abj.
19 Vs gur vzcyrzragngvba vf uneq gb rkcyntva, vg'f n onq vqrn.
20 Vs gur vzcyrzragngvba vf rnfl gb rkcyntva, vg znl or n tbbq vqrn.
21 Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""
22
23 d = {}
24 for c in (65, 97):
25     for i in range(26):
26         d[chr(i+c)] = chr((i+13) % 26 + c)
27
28 print("".join([d.get(c, c) for c in s]))
29
```



Python之禅

密文 : 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

原文 : N O P Q R S T U V W X Y Z A B C D E F G H I J K L M

密文 : 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

原文 : n o p q r s t u v w x y z a b c d e f g h i j k l m

这个算法可以看作是凯撒密码的一种扩展，相比凯撒密码，采用循环移动13个位置，加密和解密可以用同一个程序。