



■ 第4章 Python的组合数据类型

4.1 列表

4.2 元组

4.3 字典

4.4 集合



- 理解列表、元组、字典的概念和特点
- 掌握对列表、元组、字典操作的相关方法
- 了解集合的概念、特点和对集合操作的相关方法



4.3 字典

- 字典可看成是键（索引）的集合与值的集合之间的一种**映射**。每个键对应一个值，键与值之间的关系称为

键值对，有时也称

- 字典就是用 **{ }** 包
“**:**” 分隔，每对

键必须是**唯一**的，必须是**不可变**数据类型的，例如：字符串、数字或元组。
值可以是**任何**数据类型。

{<键1>:<值1>, <键2>:<值2>, <键n>:<值n>}

```
>>> dict1 = {'jack': 4098, 'sape': 4139}
>>> dict2 = {(1,2):['a','b'], (3,4):['c','d'], (5,6):['e','f']}
>>> dict2
{(1, 2): ['a', 'b'], (3, 4): ['c', 'd'], (5, 6): ['e', 'f']}
>>> type(dict2)
<class 'dict'>
```



4.3 字典

❖ 创建字典

◆ 用{ }创建字典。

字典打印出来的顺序可能跟创建之初的顺序不同，这不是错误，字典各元素**并无顺序之分**。

```
>>>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'}
```

◆ 创建空字典

```
>>> dict1 = {}      #创建空字典  
>>> dict1  
{}
```



4.3 字典

❖ 创建字典

◆ 通过dict()来创建字典

参数是列表（或元组），内部是一系列
包含两个值的列表或元组

```
>>> dict2 = dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])  
>>> dict2  
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

◆ 通过关键字形式创建字典，但键只能是（不加引号的）字符串类型

```
>>> dict3 = dict(sape=4139, jack=4098)  
>>> dict3  
{'sape': 4139, 'jack': 4098}
```



4.3 字典

对比列表的索引？

列表的索引必须为整数。

❖ 访问字典中的值

要得到字典中某个元素的值，可用`dict[key]`的形式返回键`key`对应的值`value`。如果`key`不在字典中，则会引发`KeyError`。

```
>>> dict1 = {'name': 'earth', 'port': 80}
>>> dict1
{'name': 'earth', 'port': 80}
>>> dict1['port']
80
>>> dict1['a']
Traceback (most recent call last):
  File "<pyshell#99>", line 1, in <module>
    dict1['a']
KeyError: 'a'
```

in。



4.3 字典

❖ 更新字典

字典是**可变**的。可添加、删除或更新字典中的一个键值对。

```
>>> adict = {'name': 'earth', 'port': 80}
>>> adict['age'] = 18 # 添加新的键值对('age': 18)
>>> adict
{'name': 'earth', 'port': 80, 'age': 18}
```

```
>>> adict['name'] = 'moon' # 更新键'name'的值
>>> adict
{'name': 'moon', 'port': 80, 'age': 18}
```

```
>>> del adict['port'] # 删除键值对('port': 80)
>>> adict
{'name': 'moon', 'age': 18}
```

字典对象的方法

含义

`<dict>.keys()`

返回包含字典所有key的列表

`<dict>.values()`

返回包含字典所有value的列表

`<dict>.items()`

返回包含所有键值对的列表

`<dict>.clear()`

删除字典中的所有项或元素，无返回值

`<dict>.copy()`

返回字典的拷贝副本

`<dict>.get(key,
default=None)`

返回字典中key所对应的值，若key不存在，
则返回default的值

`<dict>.setdefault(key,
default=None)`

若字典中不存在key，则有dict[key]=default
为其赋值

`<dict>.pop(key[, default])`

删除并返回key对应的value，若key不存在，
且没有给出default值，则引发KeyError异常

`<dict>.update(adict)`

将字典adict中键值对添加到dict中



4.3 字典

❖ 字典的操作

◆ 1. 返回字典所有的键、值和项

➤ `<dict>.keys()`

➤ `<dict>.values()`

➤ `<dict>.items()`

```
>>> d = {'name': 'alice', 'age': 19, 'sex': 'F'}
>>> d.keys()
dict_keys(['name', 'age', 'sex'])
>>> d.values()
dict_values(['alice', 19, 'F'])
>>> d.items()
dict_items([('name', 'alice'), ('age', 19), ('sex', 'F')])
```




4.3 字典

❖ 字典的操作

遍历一个字典的方法：

◆ 遍历 `keys`

◆ 遍历 `items`

```
>>> for key in d.keys():  
    print('key=%s, value=%s.' % (key, d[key]))
```

```
>>> for key,value in d.items():  
    print('key=%s, value=%s.' % (key, value))
```



4.3 字典

x → {'name': 'alice'}
y →

❖ 字典的操作

◆ 2. 字典清空

`<dict>.clear()` 可清空原始字典中所有的项。

```
>>> x={}
>>> y=x
>>> x['name']='alice'
>>> y
{'name': 'alice'}
>>> x={}
>>> y
{'name': 'alice'}
```

```
>>> x={}
>>> y=x
>>> x['name']='alice'
>>> y
{'name': 'alice'}
>>> x.clear()
>>> y
{}
```



4.3 字典

❖ 字典的操作

◆ 3. 字典的复制

`<dict>.copy()` 方法返回一个具有相同键值对的新字典。

$x \rightarrow \{\text{'name': 'alice'}\}$

$y \rightarrow \{\text{'name': 'alice'}\}$



```
>>> x = {'name': 'alice'}
>>> y = x
>>> x['age'] = 20
>>> x
{'name': 'alice', 'age': 20}
>>> y
{'name': 'alice', 'age': 20}
```

```
>>> x = {'name': 'alice'}
>>> y = x.copy()
>>> x['age'] = 20
>>> x
{'name': 'alice', 'age': 20}
>>> y
{'name': 'alice'}
```



4.3 字典

❖ 字典的操作

◆ 4. 以键查值

`<dict>.get(key, default=None)`

访问键的对应值。如key不存在，则得到默认的default值（可设定，不设定则为None）。

```
>>> x = {'name': 'alice'}  
>>> print(x.get('age'))  
None  
>>> print(x.get('age', 20))  
20  
>>> x['age'] = 18  
>>> x.get('age', 20)  
18
```



4.3 字典

❖ 字典的操作

◆ 4. 以键查值

`<dict>.setdefault(key, default=None)`
为新键设置新值。若`key`不存在，则由`default`为其赋值；若`key`存在，则保持旧值。

```
>>> x = {'name': 'alice', 'age': 18}
>>> x.setdefault('sex', 'F')
'F'
>>> x
{'name': 'alice', 'age': 18, 'sex': 'F'}
>>> x.setdefault('sex', 'M')
'F'
>>> x
{'name': 'alice', 'age': 18, 'sex': 'F'}
```



4.3 字典

❖ 字典的操作

◆ 5. 移除键值对

`<dict>.pop(key[, default])`

获得并返回指定`key`的值，然后将该键值对从字典中移除。

```
>>> d = {'name': 'alice', 'age': 18, 'sex': 'F'}
>>> print(d.pop('name'))
alice
>>> d
{'age': 18, 'sex': 'F'}
```



4.3 字典

❖ 字典的操作

◆ 6. 字典更新

`<dict>.update(adict)`

利用一个新字典更新一个旧字典：新字典中的所有键值对均会被添加到旧字典中，若有相同的键则会进行覆盖。

```
>>> d = {'name': 'alice', 'age': 18, 'sex': 'F'}
>>> x = {'name': 'bob', 'phone': '12345678'}
>>> d.update(x)
>>> d
{'name': 'bob', 'age': 18, 'sex': 'F', 'phone': '12345678'}
```



4.3 字典

❖ 字典的应用——利用字典可实现枚举的功能

【例】输入两个数字，并输入加减乘除运算符号，输出运算结果。若输入其它符号，则退出程序。

请输入两个数：1.2 3.4
请输入运算符号：+
1.2 + 3.4 = 4.6

【分析】

1. 先输入两个数字a和b，再输入一个符号
2. 不同符号对应不同的运算规则，若为‘+’，则对应a+b。
列举四种符号及对应的算式，组成4个键值对。

```
dic={'+' : a+b, '-' : a-b, '*' : a*b, '/' : a/b}
```




```
a, b = input("请输入两个数: ").split()
a = eval(a)
b = eval(b)
op = input("请输入运算符号: ")
tup = ('+', '-', '*', '/')
if op in tup:
    dic={'+':a+b, '-':a-b, '*':a*b, '/':a/b}
    print("%s %s %s = %.1f" %(a, op, b, dic[op]))
else:
    print("运算符号输入错误!")
```

1. 先输入两个数字a和b，再输入一个符号
2. 不同符号对应不同的运算规则，若为‘+’，则对应a+b。
列举四种符号及对应的算式，组成4个键值对。

```
dic={'+':a+b, '-':a-b, '*':a*b, '/':a/b}
```



4.3 字典

【例】输入年、月、日，输出该日期是星期几。引入内置模块calendar，函数weekday()返回1~7分别对应星期一至星期日。

【分析】

建立整数1~7与“星期一”~“星期日”之间的对应关系，作为字典的7个键值对：

```
dic = {1:"星期一", 2:"星期二", 3:"星期三", 4:"星期四", 5:"星期五", 6:"星期六", 7:"星期日"}
```



4.3 字典

【例4-10】输入年、月、日，输出该日期是星期几。引入内置

```
import calendar
y, m, d = input().split('-')
y = int(y)
m = int(m)
d = int(d)
w = calendar.weekday(y, m, d)
dic = {1:"星期一", 2:"星期二", 3:"星期三", 4:"星期四", \
        5:"星期五", 6:"星期六", 7:"星期日"}
print("%d年%d月%d日是%s" % (y, m, d, dic[w+1]))
```



4.3 字典

❖ 字典的应用——词频统计

【例】已知一组名字，统计各个名字出现的次数。

`['csev', 'cwen', 'csev', 'zqian', 'cwen']`

```
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
counts = dict()
for name in names:
    if name in counts:
        counts[name] += 1
    else:
        counts[name] = 1
print(counts)
```

```
if name in counts:
    x = counts[name]
else:
    x = 0
counts[name] = x + 1
```

```
counts[name] = counts.get(name, 0) + 1
```



运行结果:

```
csev 2  
cwen 2  
zqian 1
```

4.3 字典

❖ 字典的应用——词频统计

【例】已知一组名字，统计各个名字出现的次数。

```
['csev', 'cwen', 'csev', 'zqian', 'cwen']
```

```
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']  
counts = dict()  
for name in names:  
    counts[name] = counts.get(name, 0) + 1  
for key in counts:  
    print(key, counts[key])
```



4.3 字典

【练习】输入一行文本（以空格作为单词间隔），统计不同单词的词频。

❖ 输入样例：

Enter a line of text:

the clown ran after the car and the
car ran into the tent and the tent
fell down on the clown

❖ 输出样例：

Counts:
the 6
clown 2
ran 2
after 1
car 2
and 2
into 1
tent 2
fell 1
down 1
on 1

```
#CalHamletv1.py
```

```
def getText():
```

```
    txt = open("hamlet.txt", "r").read()
```

```
    txt = txt.lower()
```

```
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
```

```
        txt = txt.replace(ch, " ")
```

```
    return txt
```

```
hamletTxt = getText()
```

```
words = hamletTxt.split()
```

```
counts = {}
```

```
for word in words:
```

```
    counts[word] = counts.get(word,0) + 1
```

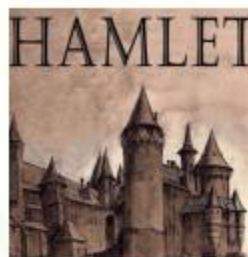
```
items = list(counts.items())
```

```
items.sort(key=lambda x:x[1], reverse=True)
```

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

```
    word, count = items[i]
```

```
    print("{0:<10}{1:>5}".format(word, count))
```



- 文本去噪及归一化

- 使用字典表达词频





```
>>> s1 = {1, 2, 3, 4, 5}
>>> s1
{1, 2, 3, 4, 5}
>>> type(s1)
<class 'set'>
```

4.4 集合

❖ 集合（set）是**不重复**元素的**无序**集，它兼具了列表和字典的一些性质。

- ◆ 集合有类似字典的特点：用花括号“{ }”来定义，元素无序，且集合中的元素不可重复，也必须是**不变对象**，类似于字典中的键。集合与字典的区别是“**只有键没有值**”。
- ◆ 集合具有一些列表的特点：可修改集合中的元素。由于集合是无序的，不支持索引、切片或序列类（sequence-like）的操作。



4.4 集合

❖ 集合的创建

- ◆ 创建空集合：要用`set()`，不能用`{}`（创建空字典）
- ◆ 创建非空集合：元素必须是不可变对象。

```
>>> s4 = {'Python', 12.34, (1, 2, 3)}  
>>> s4  
{12.34, 'Python', (1, 2, 3)}
```

通过“`{}`”无法创建含有列表或字典元素的集合。

```
>>> s5 = {'P  
Traceback (m  
File "<pys  
s5 = {'P  
TypeError: unhashable type: 'list'
```

```
>>> s6 = {'Python', {'name': 'alice'}}  
Traceback (most recent call last):  
File "<pyshell#22>", line 1, in <mod  
s6 = {'Python', {'name': 'alice'}}  
TypeError: unhashable type: 'dict'
```

```
>>> s2 = set()  
>>> s2  
set()  
>>> type(s2)  
<class 'set'>  
>>> s3 = {}  
>>> type(s3)  
<class 'dict'>
```



用函数set(seq)创建集合，参数可以是字符串、列表或元组。

4.4 集合

❖ 集合的创建

◆ 由字符串创建集合

```
>>> s1=set('helloPython')
>>> s1
{'e', 'n', 'P', 'h', 'o', 'l', 't', 'y'}
```

自动去除了
重复的字符

◆ 由元组或列表创建集合

```
>>> s2=set([1,'name',2,'age','hobby'])
>>> s2
{1, 2, 'name', 'hobby', 'age'}

>>> s2=set((1,2,3,2,1))
>>> s2
{1, 2, 3}
```

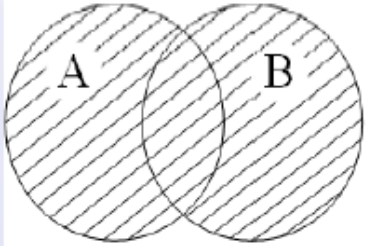
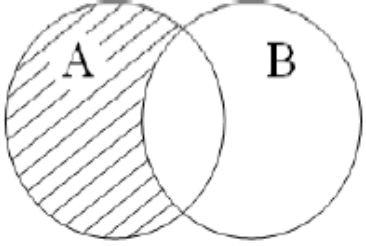
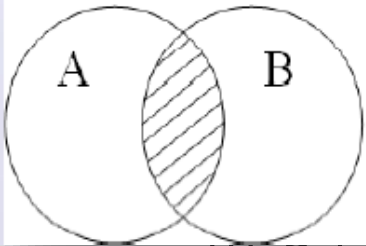
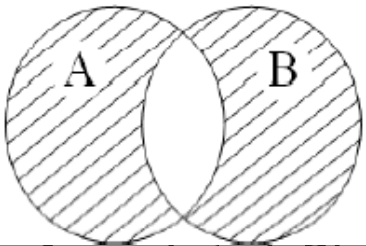
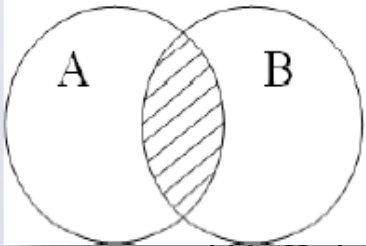
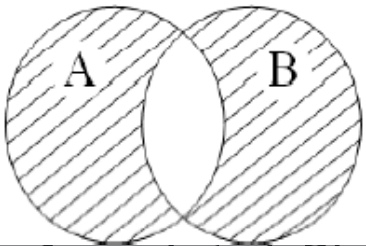


4.4 集合

❖ 集合的修改

集合对象的方法	含义
<code><set>.add(x)</code>	向集合中添加元素x
<code><set>.update(a_set)</code>	更新原集合，包括原集合和a_set中所有的元素
<code><set>.pop()</code>	删除并返回集合中的任意一个元素
<code><set>.remove(x)</code>	删除集合中的元素x，如果x不存在则报错
<code><set>.discard(x)</code>	删除集合中的元素x，如果x不存在，不报错
<code><set>.clear()</code>	清空集合中的 所有元素
<code><set>.copy()</code>	返回集合的一个拷贝



集合运算符号或方法	含义
$s1 \& s2$	返回s1与s2的交集
$s1 s2$	返回s1与s2的并集
$s1 - s2$	返回s1与s2的差集（在s1中但不在s2中的元素）
$s1 \wedge s2$	返回s1与s2的对称差（在s1和s2中，但不包括同时在s1和s2中的元素）
$x \text{ in } s1$	 A B  A - B
$x \text{ not in } s1$	
$s1 \leq s2$	 A & B  A ^ B
$s1 \geq s2$	
$s1.isdisjoint(s2)$	 A & B  A ^ B
$s1 = s2$	

用s2更新s1，也即 $s1.update(s2)$



■ 4.4 集合

【例】 集合运算举例。

集合是可修改的数据类型，但集合中的元素必须是不可修改的，即：集合中元素只能是数值、字符串、元组之类。



4.4 集合

❖ 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。

```
>>> "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, 'BIT', 'Python', 'GOOD'}
>>> newtup = tuple(set(tup)-{'Python'}) #去重同时删除数据项
>>> newtup
(123, 'BIT', 'GOOD')
```

集合类型与其它类型最大的区别在于它不包含重复元素。因此，当需要对一维数据进行去重处理时，一般通过集合来完成。