# Python程序设计

# 第五讲 批量数据类型与操作 集合与字典



张华 WHU

# 集合与字典







#### ■ 集合的特点

- \*集合是无序的,不能通过数字进行索引。
- \*集合的元素不能重复出现

#### ■ 集合的应用

- \* 去除列表中的重复元素
- \* 求两个列表的相同元素(交集)
- \* 求两个列表的不同元素(差集)

#### ■ 集合可分为两类

- \*可变集合(set):可以添加和删除元素
- \*不可变集合(frozenset):不允许添加和删除元素

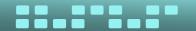


#### ■创建集合

- ♣ 使用{s1,s2,...sn}
  - ▶ 创建的是可变集合set,
  - ▶用逗号分隔的数据项作为集合的一个元素。
  - ▶因为{}表示空字典(dict),所以用set()创建空集合。

#### **\*** 举例

```
>>> s1={2,4,6,8,10}
>>> type(s1)
<class 'set'>
>>> s1
{8, 10, 4, 2, 6}
>>> s2={'hello'}
>>> s2
{'hello'}
```



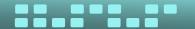
#### ■创建集合

- \*通过类型构造器set()
  - ▶ set()函数的参数是容器对象,可以是字符串,列表和元组,它可以 将序列对象的元素作为集合set的元素。

```
>>> s3=set('hello')
>>> s3
{'l', 'e', 'o', 'h'}
```

▶同样还可以根据列表对象来创建集合。

```
>>> s5=set(['he','hello','her','here'])
>>> s5
{'here', 'hello', 'he', 'her'}
```



# 集合应用

#### ■ 列表去重复操作

\*通过set函数建立列表的去重复集合元素,再通过list方法根据集合创建列表:

```
>>> L1 = [1,2,3,4,1,2,3,4]

>>> s4=set(L1)

>>> s4

{1, 2, 3, 4}

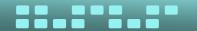
>>> L2=list(s4)

>>> L2

[1, 2, 3, 4]
```

\*L2是去重复后的列表,上面的过程也可简单地写为:

```
>>> L2=list(set(L1))
>>> print(L2)
[1, 2, 3, 4]
```



# 集合应用

#### ■ 列表去重复操作练习

- \*生成100个介于0到9999之间的随机数,找出不重复的数。 试用两种方法实现。
- \*产生指定范围内一定数量的不重复数字。

#### ■ 创建不可变集合

- \* 通过类型构造器frozenset()
  - ▶创建后的集合不能改变集合元素。
  - >例如, 创建一个星期的英文缩写的不可变集合。

```
>>> s6=frozenset(('MON','TUE','WED','THU','FRI','SAT','SUN'))
>>> s6
frozenset({'SUN', 'WED', 'TUE', 'SAT', 'FRI', 'MON', 'THU'})
```

# 集合应用

#### ■访问集合元素

- \*由于集合本身是无序的,所以不能为集合创建索引或切片操作,
- ♣ 只能循环遍历或使用in、not in来访问或判断集合元素。
- **\*** 举例

```
>>> 'SUN' in s6
True
>>> 'SON' in s6
False
>>> for day in s6:
        print(day, end=" ")
SUN WED TUE SAT FRI MON THU
```

# 操作集合

#### ■ 集合的长度、最大值、最小值、元素和

\*通过内置函数len()、max()、min()、sum(),可以获取集合的长度、元素最大值、元素最小值、元素之和。

```
>>> s7={1,2,3,4,5,6,7,8,9}
>>> len(s7)
9
>>> max(s7)
9
>>> min(s7)
1
>>> sum(s7)
45
```

#### **■** 集合支持的运算

运算	描述	运算	描述
x in <集合>	检测x是否在集合中	s1==s2	判断集合是否相等
s1 s2	并集	s1<=s2	判断s1是否是s2的子集
s1&s2	交集	s1 <s2< th=""><th>判断s1是否是s2的真子集</th></s2<>	判断s1是否是s2的真子集
s1-s2	差集	s1>=s2	判断s1是否是s2的超集
s1^s2	异或集,求s1与s2中相 异元素	s1>s2	判断s1是否是s2的真超集
s1 =s2	将s2的元素并入s1		

#### ■ 集合运算举例

```
>>> s2
{'hello'}
>>> s5
{'here', 'hello', 'he', 'her'}
            #判断s2是否是s5的子集
>>> s2<=s5
True
                    #判断s5是否是s2的真超集
>>> s5>s2
True
>>> s7={'hen','height','her'}
                #将s2并入s7
>>> s71=s2
>>> s7
{'hello', 'her', 'height', 'hen'}
                      #求s7 和s5的交集
>>> s7&s5
{'hello', 'her'}
                      #求s7 和s5的并集
>>> s7|s5
{'here', 'her', 'hello', 'hen', 'he', 'height'}
                       #求s7中去除s5中有的元素
>>> s7-s5
{'height', 'hen'}
                     #求s7 和s5的中各不相同的元素
>>> s7^s5
{'he', 'hen', 'height', 'here'}
```

#### ■ 集合运算举例

判断两个集合是否相等,只需判断其中包含的集合元素是 否一致,与顺序无关。

```
>>> s8={'he','hello', 'her','here'}
>>> s8
{'her', 'hello', 'he', 'here'}
>>> s5
{'here', 'hello', 'he', 'her'}
>>> s5==s7
True
```

#### ■ 集合运算举例

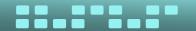
\*测试指定列表中是否包含非法数据。

```
if (set(colors)-set(lstColor)): #转换为集合之后再比较 print('error')
```

#### ■ 集合对象的方法

方法	描述
s1.union(s2)	s1 s2,返回一个新的集合对象
s1. difference(s2)	s1-s2,返回一个新的集合对象
s1. intersection(s2)	s1&s2,返回一个新的集合对象
s1. issubset(s2)	$s1 \le s2$
s1. issuperset(s2)	s1>=s2
* s1. update(s2)	将s2的元素并入s1
* s1. add (x)	增加元素x到s1
* s1. remove(x)	从s1移除x,x不存在报错
* s1. clear()	清空s1
s1. copy()	复制s1,返回一个新的集合对象
s1. union(s2)	s1 s2,返回一个新的集合对象
s1. difference (s2)	s1-s2,返回一个新的集合对象

▶打星号\*的方法是set集合独有的方法,不打星号的方法是两种集合都有的方法。



#### ■ 集合对象的方法

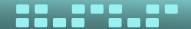
\*表中的s2并不要求是相同类型的对象,只要是一个可迭代的对象,包括字符串、列表、元组、集合。

#### ■ 集合对象的方法应用举例

- \* 利用集合分析活动投票情况:
  - >两个小队举行活动评测投票,按队员序号投票,
  - ▶第一小队队员序号为1、2、3、4、5,
  - ▶第二小队队员的序号为6、7、8、9、10,
  - >可以对投票数据进行分析,
  - ▶ 投票数据为 1,5,9,3,9,1,1,7,5,7,7,3,3,1,5,7,4,4,5,4,9,5,5,9

>>>#建立集合s2表示第一小队队员序号,s3表示第二小队队员序号:

```
>>>s2=\{1,2,3,4,5\}
```



#### ■ 集合对象的方法应用举例

```
>>>#使用投票数据建立集合s1,集合去重复后表示获得了选票的队员序号:
>>> s1={1,5,9,3,9,1,1,7,5,7,7,3,3,1,5,7,4,4,5,4,9,5,5,9}
>>> s1
\{1, 3, 4, 5, 7, 9\}
>>>#第一小队获得选票的队员有:
>>> s1-s3
{1, 3, 4, 5}
>>>#第一小队没有获得选票的队员有:
>>> s2-(s1-s3)
{2}
>>>#第二小队获得选票的队员有:
>>> s1-s2
{9, 7}
>>>#第二小队没有获得选票的队员有:
>>> s3-(s1-s2)
{8, 10, 6}
```



# 字典

#### ■ 字典是python中唯一内置映射数据类型。

- \*字典类型dict是无序的集合体。
- \* 字典是一个由键和值组成的键值对构成的集合,每一个字 典元素分为两部份:
  - ➤键(key)
  - ➤值(value)
- \*键不能重复,只能使用不可变对象(bool,int,float,complex,str, tuple, fronzenset等)。
- \* 可以通过指定的键从字典访问值。
  - ▶例如,表示星期时,用1表示星期一(MON),6表示星期六(SAT),0表示星期日(SUN)。





#### ■ 字典字面值

\*由一对花括号括起的,以逗号分隔的键值对构成,键值对的书写形式为

<键>:<值>

**\*** 举例

```
>>> d1={1:'MON',2:'TUE',3:'WED',4:'THU',5:'FRI',6:'SAT',0:'SUN'}
>>> d1
{0: 'SUN', 1: 'MON', 2: 'TUE', 3: 'WED', 4: 'THU', 5: 'FRI', 6: 'SAT'}
>>> test={"test":{"mytest":10}}
>>> test
{'test': {'mytest': 10}}
```



#### ■用类型构造器创建字典

\*参数为键值对,键值对之间以逗号分割,键值对的书写形 式为

```
<键>= <值>
```

\*举例

```
>>> monthdays = dict(
Jan=31, Feb=28,
Mar=31, Apr=30,
May=31, Jun=30,
Jul=31, Aug=31,
Sep=30, Oct=31,
Nov=30, Dec=31)
```



#### ■ 用类型构造器创建字典

- \*类型构造器对键值对的要求比字面值的键值对的要求更严格,键名key必须是一个标识符,而不能是表达式。
- **\*** 举例

```
>>> weekday=dict(1='MON',2='TUE',3='WED',4='THU',
5='FRI',6='SAT',0='SUN')
SyntaxError: keyword can't be an expression

>>> weekday=dict(a1 ='MON', a2='TUE', a3='WED', a4='THU',
a5='FRI',a6='SAT',a0='SUN')
>>> weekday
{'a3': 'WED', 'a2': 'TUE', 'a1': 'MON', 'a0': 'SUN',
'a6': 'SAT', 'a5': 'FRI', 'a4': 'THU'}
```



#### ■ 访问字典元素

\* 字典元素的访问方式是通过键访问相关联的值,访问形式为

```
<字典>[<键>]
```

\*举例

```
>>>>> d1[2]
'TUE'
>>> monthdays["Jan"]
31
>>> monthdays["Jau"]
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
KeyError: 'Jau'
```



#### ■ 字典的基本操作

- \*修改元素
  - ➤ monthdays["Jan"]=30,可把Jan的值由31改为30。
- \*添加元素
  - ➤ monthdays["test"]=30,如果不存在键"test",则添加一个新键值对。
- \*删除元素
  - ➤ del monthdays["test"],删除字典条目。





#### ■ 字典的基本操作

- \*通过内置函数len(),可以获取字典的长度(元素个数)
- \*\*字典对象也支持比较运算符(<、<=、==、!=、>=、>), 但只有==、!=有意义。







方法	描述		
d.keys()	返回字典d中所有键的列表,类型为dict_keys。		
d. values()	返回字典d中值的列表,类型为dict_values。		
d.items()	返回字典d中由键和相应值组成的元组的列表,类型为dict_items。		
d.clear()	删除字典d的所有条目。		
d. copy()	返回字典d的浅复制拷贝,不复制嵌入结构。		
d. update(x)	将字典x中的键值加入到字典d。		
d. pop(k)	删除键值为k的键值对,返回k所对应的值。		
d. get(k[, y])	返回键k对应的值,若未找到该键返回none,若提供y,则未找到k时返回y。		



#### ■ 字典对象的方法应用举例

```
>>> monthdays.keys() #显示字典monthdays的键值序列
dict keys(['Apr', 'Dec', 'May', 'Feb', 'Aug', 'Oct', 'Jan',
'Jun', 'Jul', 'Mar', 'Sep', 'Nov'])
>>> monthdays.values() #显示字典monthdays的值序列
dict values([30, 31, 31, 28, 31, 31, 30, 30, 31, 31, 30, 30])
>>>#dict keys和dict values也是一个可迭代对象,
>>>#可以通过迭代方式访问其中的元素,例如:
>>> for i in monthdays.keys():
       print(i,end=" ")
Apr Dec May Feb Aug Oct Jan Jun Jul Mar Sep Nov
>>> monthdays.items() #显示字典monthdays的键值对序列
dict items([('Apr', 30), ('Jul', 31), ('Jun', 30), ('Oct', 31),
('Mar', 31), ('Jan', 30), ('May', 31), ('Nov', 30), ('Dec', 31),
('Aug', 31), ('Sep', 30), ('Feb', 28)])
```



#### ■ 字典对象的方法应用举例

```
>> x={'a1':21,'a2':34} #创建一个新的字典x
>>> x
{'a2': 34, 'a1': 21}
>>> monthdays.update(x) #将字典x的键值对追加到字典monthdays中
>>> monthdays
{'Apr': 30, 'Jul': 31, 'Jun': 30, 'Oct': 31, 'Mar': 31, 'Jan': 30,
'May': 31, 'Nov': 30, 'Dec': 31, 'a2': 34, 'a1': 21, 'Aug': 31,
'Sep': 30, 'Feb': 28}
>>> monthdays.pop('a1') #删除键为'a1'的键值对
21
>>> monthdays
{'Apr': 30, 'Jul': 31, 'Jun': 30, 'Oct': 31, 'Mar': 31, 'Jan': 30,
'May': 31, 'Nov': 30, 'Dec': 31, 'a2': 34, 'Aug': 31, 'Sep': 30,
'Feb': 28}
>>> monthdays.get('a2') #获取键'a2'对应的值
34
>>> monthdays.get('a1','not found') #没有找到'a1'则返回'not found'
'not found'
```

# 字典应用

#### ■ 字典应用举例

♣ 已有5位同学的姓名和成绩,按成绩从高到低列出同学姓名 ,假设成绩没有重复值。

#### \*方法一

- ▶按<成绩:姓名>建立字典,从字典获取由成绩组成的列表,
- ▶从高到低排序后,根据列表中的成绩,逐个从字典中查找对应的姓名,写入另一个列表。
- >得到的新列表中的姓名就是按成绩排序的。



# 字典应用

#### ■ 字典应用举例

```
>>> scores={85:"李鸣",74:"黄辉",92:"张檬",88:"于静颂",63:"钱多多"}
>>> scores
{88: '于静颂', 74: '黄辉', 92: '张檬', 85: '李鸣', 63: '钱多多'}
>>> L1=list(scores.keys())
>>> T.1
[88, 74, 92, 85, 63]
>>> L1.sort(reverse=True)
>>> L1
[92, 88, 85, 74, 63]
>>> L2=[]
>>> for i in range(0,len(L1)):
        L2.append(scores[L1[i]])
>>> T.2
['张檬', '于静颂', '李鸣', '黄辉', '钱多多']
```

# 字典应用

#### ■ 字典应用举例

\*思考:如果有相同成绩,怎么操作呢?

```
>>> scores={"李鸣":85,"黄辉":74,"张檬":92,"于静颂":88,"钱多多":74}
>>> s1=[(v,k) for k,v in scores.items()]
>>> s1.sort(reverse=True)
>>> L1=[s1[i][1] for i in range(len(s1))]
>>> L1
['张檬', '于静颂', '李鸣', '黄辉', '钱多多']
```

# 字典应用举例

#### ■ 问题: 简单推荐系统

\*假设已有若干用户名字及其喜欢的电影清单,现有某用户 ,已看过并喜欢一些电影,现在想找个新电影看看,又不 知道看什么好。

#### ■ 设计思路

根据已有数据,查找与该用户爱好最相似的用户,也就是 看过并喜欢的电影与该用户最接近,然后从那个用户喜欢 的电影中选取一个当前用户还没看过的电影,进行推荐。



# 字典应用举例

#### ■ 实现

```
user0:{'film4', 'film9'}
user1:{'film6', 'film4', 'film2', 'film5', 'film7'}
user2:{'film6', 'film1'}
user3:{'film4', 'film2', 'film1', 'film3', 'film8', 'film7'}
user4:{'film6', 'film1', 'film5', 'film3'}
user5:{'film9'}
user6:{'film6', 'film4', 'film2', 'film5', 'film1', 'film8', 'film7'}
user7:{'film9', 'film4', 'film1', 'film3', 'film7'}
user8:{'film6', 'film4', 'film9', 'film1', 'film7'}
user9:{'film8', 'film5', 'film4', 'film9'}
```

# 字典应用举例

#### ■ 实现

```
和您最相似的用户是: user3
Ta最喜欢看的电影是: {'film4', 'film2', 'film1', 'film3', 'film8', 'film7'}
Ta看过的电影中您还没看过的有: {'film8', 'film7', 'film4'}
```



# lambda表达式

#### ■ lamda表达式

#### lambda arg1[,arg2,...]:<expression>

- \*用来声明匿名函数,也就是没有函数名字的临时使用的小函数,尤其适合需要一个函数作为另一个函数参数的场合。
- \* lambda表达式只可以包含一个表达式,该表达式的计算结果可以看作是函数的返回值,不允许包含复合语句,但在表达式中可以调用其他函数。

# 标准库collections中与字典有关的类

#### **■ OrderedDict**类

- \* Python内置字典dict是无序的,如果需要一个可以记住元素插入顺序的字典,可以使用collections.OrderedDict。
- **\*** 举例

```
>>> import collections
>>> x = collections.OrderedDict() #有序字典
>>> x['a'] = 3
>>> x['b'] = 5
>>> x['c'] = 8
>>> x
OrderedDict([('a', 3), ('b', 5), ('c', 8)])
```

# 标准库collections中与字典有关的类

#### **■ defaultdict类**

- \* 可以指定字典中值的类型。
- **\*** 举例

```
>>> import string
>>> import random
>>> x = string.ascii_letters + string.digits + string.punctuation
>>> y = [random.choice(x) for i in range(1000)]
>>> z = ''.join(y)
>>> from collections import defaultdict
>>> frequences = defaultdict(int) #所有值默认为0
>>> frequences
defaultdict(<class 'int'>, {})
>>> for item in z:
    frequences[item] += 1 #修改每个字符的频次
>>> frequences.items()
```

#### COMPUTER PROGRAMMING

# 标准库collections中与字典有关的类

#### **■ Counter**类

\*\*对于频次统计的问题,使用collections模块的Counter类可以更加快速地实现这个功能,并且能够提供更多的功能,例如查找出现次数最多的元素。

#### \*举例

```
>>> import string
>>> import random
>>> x = string.ascii_letters + string.digits + string.punctuation
>>> y = [random.choice(x) for i in range(1000)]
>>> z = ''.join(y)
>>> from collections import Counter
>>> frequences = Counter(z)
>>> frequences.items()
>>> frequences.most_common(1) #返回出现次数最多的1个字符及其频率
>>> frequences.most_common(3) #返回出现次数最多的前3个字符及其频率
```



### 小结

#### **■** Python的批量数据类型

