Python脚本语言

第四周 组合数据结构

陈世峰 岭南师范学院信息工程学院
chenshifeng@lingnan.edu.cn/13234075348

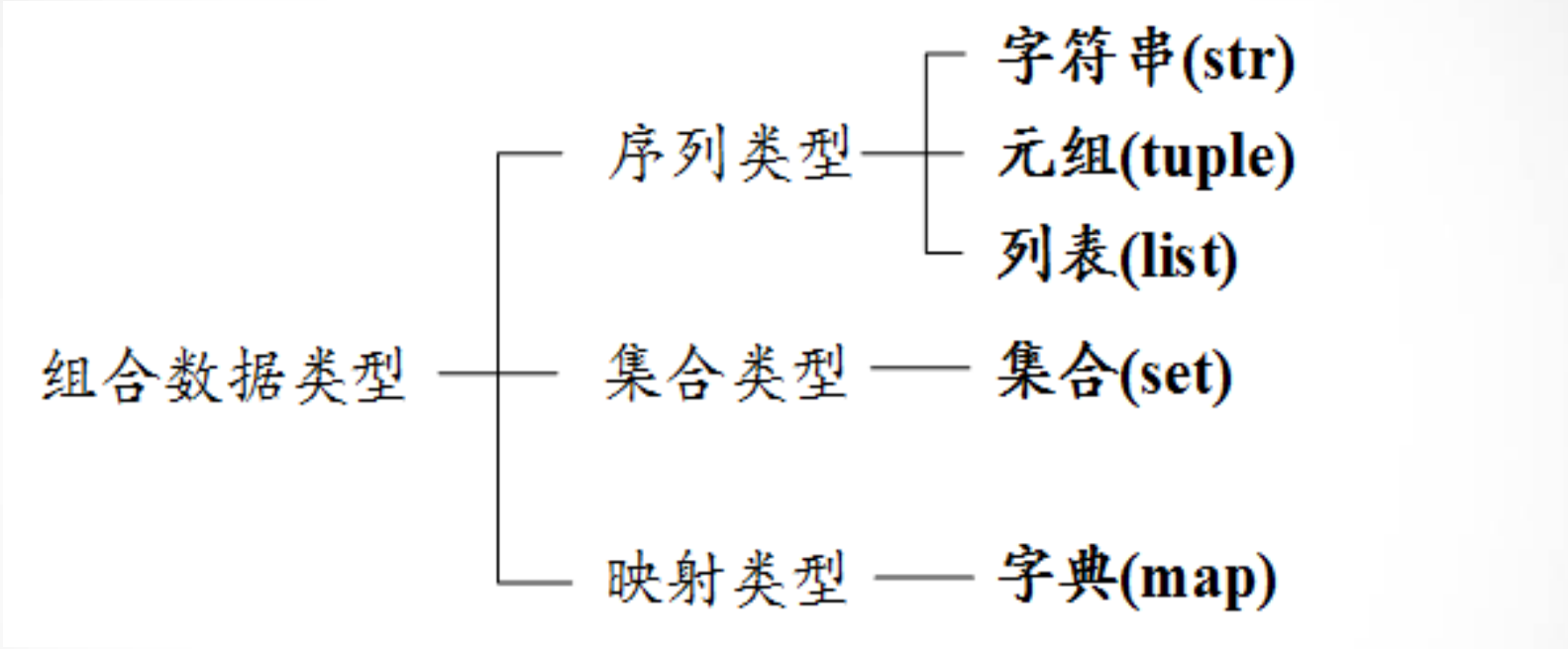
引言

为了在计算机程序中表示现实世界中更加复杂的数据，Python除了提供数字和字符串等数据类型，还提供了元组、列表、字典和集合等复杂类型的数据结构。

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，通过单一的表示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。

教学目的

- 掌握列表结构的操作方法
- 掌握元组的常见操作方法
- 掌握字典数据结构的操作方法
- 掌握集合数据结构的操作方法



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。

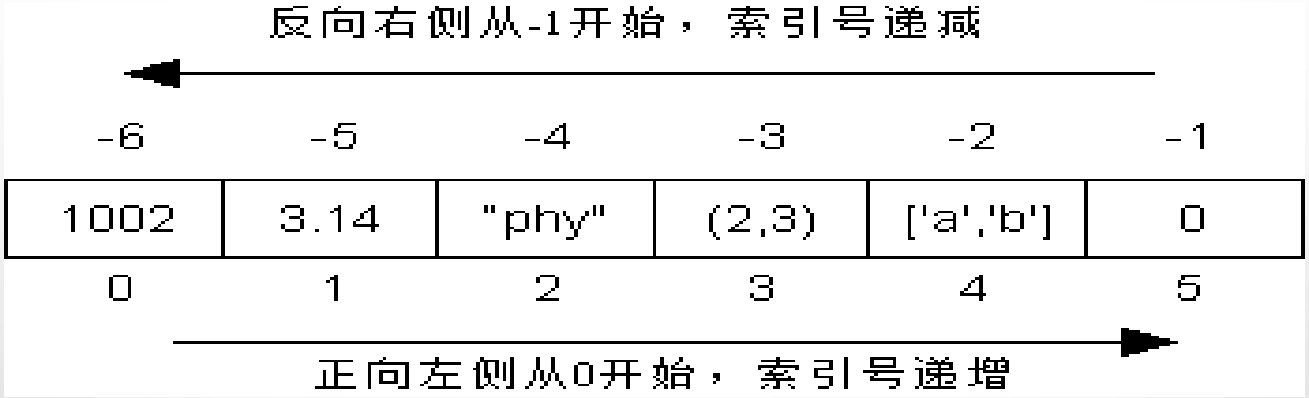
集合类型是一个元素集合，元素之间无序，相同元素在集合中唯一存在。

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

序列是Python中最基本的数据结构，其中最常见的就是元组、列表和字符串。

- 序列类型的元素之间存在先后关系，可以通过索引来访问。
- 序列类型支持成员关系操作符(in)、分片运算符([])，序列中的元素也还可以是序列类型。
- 单一字符串只表达一个含义，也被看作是基本数据类型。
- 序列类型，都可以使用相同的索引体系，即正向递增序号和反向递减序号



序列常见操作方法

操作符或函数	功能描述
x in s	如果x是s的元素返回True，否则返回 False
x not in s	如果x不是s的元素返回True，否则返回 False
s+t	返回s和t的连接
s*n	将序列s复制n次
s[i]	索引，返回序列的第i个元素
s[i:j]	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)
s[i:j:k]	返回包含序列s第i到j个元素以k为步长的子序列
len(s)	返回序列s的元素个数(长度)
min(s)	返回序列s中的最小元素
max(s)	返回序列s中的最大元素
s.index(x[,i[,j]])	序列s中从i开始到j位置中第一次出现元素x的位置
s.count(x)	序列s中出现x的总次数

元组 Tuple

- 是包含0个或多个元素的不可变序列类型。
- 任何元素不能替换或删除。
- 元组元素可以存储不同类型的数据，可以是字符串、数字，甚至是元组。
- 元组用小括号（）表示

```
>>> t1=('python',2019,'soft',2020)
>>> t2=(1,2,3,4,5)
>>> t3="a","b","c","d" #括号可以省略
>>> t4=(1,) #元组只有一个元素时，逗号不可省略
>>> t5=((1,2,3),(4,5),('python','c#','Java'),6)
>>> t6=()
>>> t7=tuple()
```

元组的常见操作

（1）访问元组元素，方法：

元组[索引]

（2）获取元组长度，即元组中元素的数量。

可以通过len(元组名)获取元组的长度。

（3）遍历元组元素

可以使用for语句和range()函数遍历列表索引，然后通过索引依次访问每个元组元素，方法：

```
for i in range(len(tuple)):
    访问tuple[i]
```

也可以使用for语句和enumerate()函数同时遍历元组的元素索引和元素值，方法：

```
for 索引,元素值 in enumerate(tuple):
    访问索引和元素值
```

```
>>> tuple = ('王二','张三','李四','王五')
>>> tuple[0]
'王二'
>>> tuple[-1]
'王五'
>>> len(tuple)
4
>>> for i in range(len(tuple)):
    print(tuple[i])
王二
张三
李四
王五
>>> for index,value in enumerate(tuple):
    print("第%d个元素值是【%s】" %(index,value))
第0个元素值是【王二】
第1个元素值是【张三】
第2个元素值是【李四】
第3个元素值是【王五】
```

列表

列表（List）是一组有序存储的数据。例如，饭店点餐的菜单就是一种列表。列表具有如下特性：

- 和变量一样，每个列表都有一个唯一标识它的名称。
- 列表中**不同元素的类型可以相同，也可以不同**。它支持数字，字符串甚至可以包含列表（所谓嵌套）。
- 每个列表元素都有索引和值两个属性，索引是一个从0开始的整数，用于标识元素在列表中的位置；值就是元素对应的值。
- 列表常量用**[ ]**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。

5.2 列表

- **列表的方法**

列表还有特有的函数或方法用于完成列表元素的增删改查，其中ls、lst分别为两个列表，x是列表中的元素，i和j是列表的索引。

操作符	操作符
ls[i]=x	ls.append(x)
ls[i:j]=lst	ls.clear()
ls[i:j:k]=lst	ls.copy()
del ls[i:j]	ls.insert(i,x)
del ls[i:j:k]	ls.pop(i)
ls+=lst(ls.extend(lst))	ls.remove(x)
ls*=n	ls.reverse(x)
	ls.sort()

创建列表

- 创建一个列表，使用方括号[ ]将用逗号分隔的元素括起来即可：

```
>>> list1 = [1,2,3,4,5]
>>> list1
[1, 2, 3, 4, 5]
```

- 列表中的元素也可以是不同的数据类型：

```
>>> list2 = [1,2,3,'a','b','c']
>>> list2
[1, 2, 3, 'a', 'b', 'c']
```

访问列表中的数据

- 和元组一样，我们也可以使用索引或分片访问列表中的元素：

```
>>> list2[3]
'a'
>>> list2[3:6]
['a', 'b', 'c']
```

列表元素赋值

- 我们可以将列表数据通过赋值存放到单个变量中，然后通过索引值对列表特定位置元素进行赋值：

```
>>> list2 = [1,2,3,'a','b','c']
>>> list2
[1, 2, 3, 'a', 'b', 'c']
>>> list2[2] = 4                #将列表中数字3用数字4替换
>>> list2
[1, 2, 4, 'a', 'b', 'c']
>>> list2[5] = 'd'              #将字符c用字符d替换
>>> list2
[1, 2, 4, 'a', 'b', 'd']
```

- 还可以通过分片将列表中的一部分元素赋值给新的变量：

```
>>> char = ['a','b','e','f']
>>> char[2:] = ['c','d']
>>> char
['a', 'b', 'c', 'd']
```

- 除了普通的赋值，列表分片赋值语句可以在不需要替换任何原有元素的情况下插入新的元素：

```
>>> number = [1,5,6]
>>> number[1:1] = [2,3,4]
>>> number
[1, 2, 3, 4, 5, 6]
```

- 以此类推，通过分片赋值还可以实现删除列表中元素的操作：

```
>>> number = [1, 2, 3, 4, 5, 6]
>>> number[1:4] = [] #结果和del number[1:4]相同
>>> number
[1, 5, 6]
```

删除列表中的元素

- 有时，我们需要删除列表中的数据，我们可以使用del命令来完成，例如：

```
>>> list2 = [1,2,3,'a','b','c']
>>> del list2[5]
>>> list2
[1, 2, 3, 'a', 'b']
```

列表的append方法

- 如果你想往现有的列表中追加新的元素，可以通过使用列表的append方法，例如：

```
>>> name = ['Zhao','Qian','Sun','Li']
>>> name.append('Zhou')
>>> name
['Zhao', 'Qian', 'Sun', 'Li', 'Zhou']
>>> name.append(‘Wu’,‘Zheng’)
#使用append方法，每次只能在列表末尾追加一个元素
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    name.append('Zhou','Wu')
TypeError: append() takes exactly one argument (2 given)
```

列表的extend 方法

- 通过列表的extend方法可以将新列表扩展到原有列表中，例如：

```
>>> name = ['Zhao', 'Qian', 'Sun', 'Li']
>>> name1 = ['Zhou','Wu','Zheng','Wang']
>>> name.extend(name1)
>>> name
['Zhao', 'Qian', 'Sun', 'Li', 'Zhou', 'Wu', 'Zheng', 'Wang']
```

列表的insert方法

- 通过列表的insert方法，可以将新元素插入到列表指定位置：

```
>>> number = [1,2,3,5,6,7,8]
>>> number.insert(3,4) #第一个值3是索引，第二个值4为要插入的值
>>> number
[1, 2, 3, 4, 5, 6, 7, 8]
```

列表的index方法

列表中的index方法可以找出某个值的第一个匹配项的索引值，如果列表中不包含你要找的数据，Python会给出相应的报错信息，例如：

```
>>> name = ['Zhao','Qian','Sun','Li','Zhao']
>>> name.index('Zhao')
0
>>> name.index('Wang')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    name.index('Wang')
ValueError: 'Wang' is not in list
```

列表的count方法

列表中的count方法可以用来统计某个元素在列表中出现次数：

```
>>> list3 = ['a','b','c','d','e','f','e']
>>> list3.count('e')
2
```

列表的pop方法

- 列表中pop方法可以移除列表中的某个元素（默认是最后一个元素），并返回该元素的值：

```
>>> number = [1,2,3]
>>> number.pop()
3
>>> number
[1, 2]
>>> number.pop(0)
1
>>> number
[2]
```

列表的remove方法

- 列表中的remove方法可以用来移除列表中的某个元素的第一个匹配项，与index方法一样，如果没有找到相应的元素，Python则会产生报错，例如：

```
>>> char = ['a','b','c','a']
>>> char.remove('a')
>>> char
['b', 'c', 'a']
>>> char.remove('d')
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    char.remove('d')
ValueError: list.remove(x): x not in list
```

列表的reverse方法

列表reverse方法将列表中的元素反向放置，这个操作也被称为逆置：

```
>>> number = [1,2,3,4,5,6]
>>> number.reverse()
>>> number
[6, 5, 4, 3, 2, 1]
```

列表的sort方法

列表的sort方法可以对列表进行排序，默认的排序方式为从小到大：

```
>>> number = [3,2,1,5,4,6]
>>> number.sort()
>>> number
[1, 2, 3, 4, 5, 6]
```

遍历列表元素

遍历列表就是一个一个地访问列表元素，可以使用for语句和range()函数遍历列表索引，然后通过索引依次访问每个列表元素，方法如下：

```
for i in range(len(list)):
    访问list[i]
```

也可以使用for语句和enumerate()函数同时遍历列表的元素索引和元素值，方法如下：

```
for 索引,元素值 in enumerate(list):
    访问索引和元素值
```

推导式

列表推导式是一种可以快速生成List的方法

```
list1 = []
for i in range(101):
    if i%2 == 0:
        list1.append(i)
```

```
list1 = [i for i in range(101) if i%2 == 0]
```

其中最开始的i是代表元素，

for i in range(101) 说明了这个元素的取值范围，

最后一个if 是限制条件。

```
list1 = [i*i for i in range(10)]
```

多维列表

可以将多维列表视为列表的嵌套。即多维列表的元素值也是一个列表，只是维度比其父列表小1。二维列表的元素值是一维列表，三维列表的元素值是二维列表，以此类推。

【例2-37】一个定义二维列表的例子。

```
list2 = [["CPU","内存"],["硬盘","声卡"]]
```

打印二维列表

```
list2 = [["CPU","内存"],["硬盘","声卡"]]
for i in range(len(list2)):
    print(list2[i])
```

运行结果如下：

```
['CPU','内存']
['硬盘','声卡']
```

获取二维列表元素的值

可以使用下面的方法获取二维列表元素的值：

列表名[索引1][索引2]

```
list2 = [["CPU","内存"],["硬盘","声卡"]]
for i in range(len(list2)):
    for j in range(len(list2[i])):
        print(list2[i][j])
```

运行结果如下：

```
CPU
内存
硬盘
声卡
```

元组与列表的区别

- 元组中的数据一旦定义就不允许更改。
- 元组没有append()、extend()和insert()等方法，无法向元组中添加元素。
- 元组没有remove()或pop()方法，也无法对元组元素进行del操作，不能从元组中删除元素。
- 从效果上看，tuple( )冻结列表，而list( )融化元组。

字典

- 字典是一种通过名字来引用值的数据结构。这种类型的数据结构称为映射 (Mapping)。字典是Python中唯一内建的映射类型。字典中的值(value)没有特殊的顺序，但是都存储在特定的键(key)下。

- 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。

字典的创建

字典由多个（键：值）对组成，键和值之间通过冒号分割，所有的键值对用大括号括起来，键值对之间使用逗号分割。

```
>>> d = {'a':1,'b':2,'C':3,'d':4,'e’:5}
```

也可以通过dict()函数创建一个空字典：

```
>>> dict()
{}
```

访问字典中的数据

和访问序列元素的方式相似，通过在中括号中填入键名对字典中该键所对应的值进行访问，例如：

```
>>> d['a']
1
```

若索访问的键值对不存在就会输出错误：

```
>>> d['f']
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    d['f']
KeyError: 'f'
```

更新字典中的数据

向字典中添加新内容的方法是增加新的（键：值）对：

```
>>> d = {'a':1,'b':2,'c':3,'d':4,'e':4}
>>> d['f'] = 5
>>> d
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 4, 'f': 5}
```

使用del命令删除字典中的指定键值对：

```
>>> del d['f']
>>> d
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 4}
```

清空字典

与del命令不同，使用clear()方法可以清空字典中的所有项，使字典变为空字典：

```
>>> d = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 4}
>>> d.clear()
>>> d
{}
```

字典的copy方法

字典的copy方法返回一个具有相同键-值对的新字典（这个方法实现的是浅复制，因为值本身是相同的，而不是副本）：

```
>>> x = {'a':1,'b':[2,3,4]}
>>> y = x.copy()
>>> y['a'] = 5
>>> y['b'].remove(3)
>>> y
 {'a':5,'b':[2,4]}
>>> x
 {'a':1,'b':[2,4]}
```

字典的fromkeys方法

字典的fromkeys方法使用给定的键建立新的字典，每个键默认对应的值为None，可以直接在所有字典的类型dict上调用此方法。如果不想使用默认值，也可以自己提供值：

```
>>> {}.fromkeys(['name','age'])
{'age':None,'name':None}
>>> dict.fromkeys(['name','age'],'unknow')
{'age':'unknow','name':'unknow'}
```

字典的get方法

字典的get方法是个更宽松的访问字典项的方法。当使用get访问一个不存在的键时，会得到None值。还可以自定义“默认”值，替换None：

```
>>> d = {}
>>> print d.get('name')
None
>>> d.get("name",'N/A')
'N/A'
>>> d["name] = 'Eric'
>>> d.get('name')
'Eric'
```

字典的pop方法

- 字典的pop方法用来获得对应给定键的值，然后将这个键-值对从字典中移除：

```
>>> d = {'a':1,'b':2,'c':3}
>>> d.pop('a')
>>> d
{'b':2,'c':3}
```

字典的setdefault方法

- 字典的setdefault方法在某种程度上类似于get方法，就是能够获得与给定键相关联的值，还能在字典中不含有给定键的情况下设定相应的键值：

```
>>> d = {}
>>> d.setdefault('name','N/A')
'N/A'
>>> d
{'name': 'N/A'}
>>> d.setdefault('name','New Value')
'N/A'
```

字典的update方法

- 字典update方法可以利用一个字典项更新另一个字典。提供的字典项会被添加到旧的字典中，若有相同的键则会进行覆盖：

```
>>> d = {'a':1,'b':2,'c':3}
>>> x = {'a':5,'d':6}
>>> d.update(x)
>>> d
{'a': 5, 'c': 3, 'b': 2, 'd': 6}
```

字典的values方法

- 字典values方法以列表的形式返回字典中的值（itervalues返回值的迭代器），与返回键的列表不同的是，返回值列表中可以包含重复的元素：

```
>>> d = {1: 1, 2: 2, 3: 3, 4: 1}
>>> d.values()
dict_values[1, 2, 3, 1]
```

2.3.2 字典元素的读取

```
aDict={'name':'Dong', 'sex':'male', 'age':37}
for item in aDict.items():          #输出字典中所有元素
    print(item)
('age', 37)
('name', 'Dong')
('sex', 'male')

for key in aDict:                   #不加特殊说明，默认输出键
  print(key)
age
name
sex
```

2.3.2 字典元素的读取

```
aDict={'name':'Dong', 'sex':'male', 'age':37}

for key, value in aDict.items():          #序列解包用法
    print(key, value)
age 37
name Dong
sex male

aDict.keys()                              #返回所有键
dict_keys(['name', 'sex', 'age'])

aDict.values()                            #返回所有值
dict_values(['Dong', 'male', 37])
```

字典嵌套

有的时候，我们还需要把字典中的元素设置为另一个字典数据，从而创建一个嵌套字典，例如：

```
>>>peopleInfor = {'Tom':{'phonenumber':'2354','addr':'street4'},
'Suan':{'phonenumber':'9123','addr':'street7'},
'Nick':{'phonenumber':'5789','addr':'street10'}}
>>>peopleInfor['Tom']
>>>{'addr': 'street4', 'phonenumber': '2354'}
>>>peopleInfor['Tom']['phonenumber']
>>> '2354'
```

集合

- Python的集合(set)和数学中的定义保持一致, 是一个无序不重复元素集。
- 集合(set)是0个或多个元素的无序组合，但集合本身是可变的.
- 集合中的元素只能整数、浮点数、字符串等基本数据类型
- 集合中的元素是无序的，没有索引位置的概念。
- 集合中的任何元素都没有重复

创建集合

Python中使用set()函数创建集合：

```
>>> set('123456')
{'2', '3', '4', '1', '5', '6'}
```

集合的一个重要特点就是内部元素不允许重复：

```
>>> set('aabbccddee')
{'c', 'a', 'd', 'e', 'b'}
```

集合数据的添加

- Python中集合的添加有两种方法，分别是add和update。
- 集合add方法：

```
>>> a = set('I Love')
>>> a.add('python')
>>> a
{'o', 'v', 'I', 'e', 'python', ' ', 'L'}
```

- 集合update方法：是把要传入的元素拆分，做为个体传入到集合中，例如：

```
>>> a = set('I Love')
>>> a.update('python')
>>> a
{'h', 'o', 'p', 'v', 't', 'I', 'e', 'n', 'y', ' ', 'L'}
```

集合元素的删除

- 集合删除操作方法：remove

```
>>> a = set('12345')
>>> a.remove('5')
>>> a
{'1', '4', '3', '2'}
```

子集和超集

- 对于两个集合A与B，如果集合A的任何一个元素都是集合B的元素，我们就说集合A包含于集合B，或集合B包含集合A，也就是说集合A是集合B的子集。如果集合A是集合B的子集，则称集合B是集合A的超集。
- 如果集合A的任何一个元素都是集合B的元素，而集合B中至少有一个元素不属于集合A，则称集合A是集合B的真子集。
- 空集是任何集合的子集。任何一个集合是它本身的子集，空集是任何非空集合的真子集。

集合的数学运算

数学符号	含义	Python运算符	集合的方法
$A - B$	A和B的差集	A-B	A.difference(B)
$A \cap B$	A和B的交集	A&B	A.intersection(B)
$A \cup B$	A和B的并集	A|B	A.union(B)
$A \neq B$	A不等于B	A!=B	
$A = B$	A等于B	A==B	
$a \in A$	a属于A	a in A	
$a \notin A$	a不属于A	a not in A	
$A \subseteq B$	A是B的子集	A<=B	A.issubset(B)
$A \subset B$	A是B的真子集	A<B	
$A \supseteq B$	B是A的子集	A>=B	A.issuperset(B)
$A \supset B$	B是A的真子集	A>B	

2.4.2 集合操作

- Python集合支持交集、并集、差集等运算

```
>>> a_set = set([8, 9, 10, 11, 12, 13])
>>> b_set = {0, 1, 2, 3, 7, 8}
>>> a_set | b_set                    #并集
{0, 1, 2, 3, 7, 8, 9, 10, 11, 12, 13}
>>> a_set.union(b_set)               #并集
{0, 1, 2, 3, 7, 8, 9, 10, 11, 12, 13}
>>> a_set & b_set                    #交集
{8}
>>> a_set.intersection(b_set)        #交集
{8}
>>> a_set.difference(b_set)          #差集
{9, 10, 11, 12, 13}
>>> a_set - b_set
{9, 10, 11, 12, 13}
```

2.4.2 集合操作

```
>>> a_set.symmetric_difference(b_set)    #对称差集
{0, 1, 2, 3, 7, 9, 10, 11, 12, 13}
>>> a_set ^ b_set
{0, 1, 2, 3, 7, 9, 10, 11, 12, 13}
>>> x = {1, 2, 3}
>>> y = {1, 2, 5}
>>> z = {1, 2, 3, 4}
>>> x.issubset(y)                          #测试是否为子集
False
>>> x.issubset(z)
True
>>> {3} & {4}
set()
>>> {3}.isdisjoint({4})                    #如果两个集合的交集为空，返回
True
```

2.4.2 集合操作

- 集合包含关系测试

```
>>> x = {1, 2, 3}
>>> y = {1, 2, 5}
>>> z = {1, 2, 3, 4}
>>> x < y                                  #比较集合大小/包含关系
False
>>> x < z                                  #真子集
True
>>> y < z
False
>>> {1, 2, 3} <= {1, 2, 3}                 #子集
True
```

可变数据类型和不可变数据类型

以下所有的内容都是基于内存地址来说的。

不可变数据类型： 当该数据类型的对应变量的值发生了改变，那么它对应的内存地址也会发生改变，对于这种数据类型，就称不可变数据类型。

可变数据类型 ：当该数据类型的对应变量的值发生了改变，那么它对应的内存地址不发生改变，对于这种数据类型，就称可变数据类型。

总结：不可变数据类型更改后地址发生改变，可变数据类型更改地址不发生改变

- 可变数据类型: 列表、字典、集合
- 不可变数据类型: 数字、字符串、元组

```
>>> l=[1,2,3,4]
>>> id(l)
2459321268744
>>> l[1]=5
>>> l
[1, 5, 3, 4]
>>> id(l)
2459321268744
>>> a=1
>>> id(a)
140720095355728
>>> a=3
>>> a
3
>>> id(a)
140720095355792
>>> print("数字是不可变类型")
数字是不可变类型
```

```
>>> s = 'hello'
>>> s[1]=a
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    s[1]=a
TypeError: 'str' object does not support
item assignment
>>> id(s)
2459324230600
>>> s+='world'
>>> s
'helloworld'
>>> id(s)
2459324222576
>>> print("字符串是不可变类型")
字符串是不可变类型
```

本章小结

- 序列是Python中最基本的数据结构，其中最常见的就是元组、列表和字符串。
- 元组是一种序列，就像列表一样。元组和列表之间的主要区别是元组不能像列表那样改变元素的值，可以简单地理解为“只读列表”。
- 元组使用小括号()将数据包含起来，而列表使用方括号[]。

本章小结

- 列表中的数据可以进行任意地添加、修改和删除，
- 同时我们还可以使用index或者find函数进行元素的查找；
- 为了让列表保存复杂而庞大的数据内容，有的时候我们还会将列表嵌套使用，即列表中包含的元素也是列表数据类型的。

本章小结

- 除了元组和列表，字符串也是序列类型数据之一，字符串或串(String) 是由数字、字母、下划线组成的一串字符，用一对引号包含。它是编程语言中表示文本的数据类型。对于序列类型的数据，必须掌握对齐进行切片操作的正确方法，当方括号中的序号为正数时代表从左往右计数，而序号为负数时，则表示该序号为从序列的尾部向头部计数得到的值，即从右往左计数的值。

本章小结

- 字典是一种通过名字来引用值的数据结构。这种类型的数据结构称为映射 (Mapping)。
- 字典是Python中唯一内建的映射类型。字典中的值(value)没有特殊的顺序，但是都存储在特定的键(key)下。
- 和列表类似，字典中的数据也可以进行任意的添加、修改、删除，唯一不同的是我们无法使用一个序号来表示字典中的数据，而是使用键的定义。

本章小结

- Python的集合(set)是一个无序不重复元素集，它的操作与数学中的操作方式保持一致，通过集合运算符，你也可以完成并、交、差等集合运算。