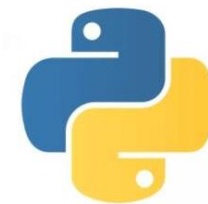


组合数据类型





内容

- Python中变量的存储
- 列表类型
- 元组类型
- 集合类型
- 字典类型



Python中变量的存储

Python中所有的变量都是指针

- Python中所有可赋值的东西，即所有可出现在赋值语句中“=”左侧的东西都是指针
- 指针指向内存单元中的地址
- 对变量进行赋值的本质，即是让该变量指向某内存地址，该地址中存储着变量的值



Python中变量的存储

- 对变量进行赋值，即让变量指向某地址

`a = 3`

`a` → 3

`b = 4`

`b` → 4

- 将某变量赋值为另一变量的本质，是让该变量指向另一变量指向的地址

`b = a`

`a` → 3
`b` → 3

`b == a`

`b is a`



Python中变量值的修改

- 对变量的值进行修改，可能的方式有两种

`a = 3` `a` → 3
4

`a = 4` `a` → 3
4

`a = 4` `a` → 34
4

修改变量指针指向的地址，在新地址中存储新的值。即，改变存储变量值的地址

修改变量指针当前指向地址中存储的值。即，不改变存储变量值的地址



不可变数据类型和可变数据类型

根据变量所指向内存地址中的值是否可以改变，将数据类型分为两种

- 不可变数据类型：变量所指向内存地址中的值不能发生改变。即，对于此类变量，如果值发生改变，则内存地址发生改变。数字、布尔、字符串及元组类型是不可变数据类型
- 可变数据类型：变量所指向内存地址中的值可以发生改变。即，对于此类变量，如果值发生改变，内存地址一般不发生改变。集合、列表、字典是可变数据类型

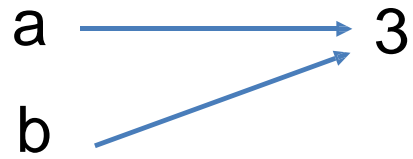


不可变数据类型变量不会相互影响

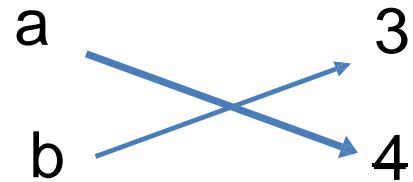
`a = 3`



`b = a`



`a = 4`





可变数据类型中可能存在的“陷阱”

`x = [1, 2, 3]`

`x` → [1, 2, 3]

`y = x`

`x` → [1, 2, 3]
`y` → [1, 2, 3]

`x[0] = 8`

`x` → [8, 2, 3]
`y` → [8, 2, 3]

```
>>> x = [1, 2, 3]
>>> y = x
>>> y
[1, 2, 3]
>>> x[0] = 8
>>> x
[8, 2, 3]
>>> y
[8, 2, 3]
```




数据类型及其分类

数据类型		类型名称	示例	基本数据类型
不可变数据类型	数字	int float complex	1234 3.14, 1.2e5 5+8j	
	布尔型	bool	True, False	
	字符串	str	'hello world' "C" """这是一个字符串"""	
可变数据类型	元组	tuple	(3, -5, 8)	组合数据类型
	列表	list	[1,2,3] ['a', 'b', 'c']	
	集合	set	{-5, 0, 'a'}	
	字典	dict	{1:"金牌", 2:"银牌", 3:"铜牌"}	



序列类型的共同特点

- 每个元素按照其在序列中的顺序都有相应的序号，代表它所在的位置。元素的序号有两个，一个正向，从0开始，依次加1；一个逆向，从-1开始，依次减1
- 可以使用元素序号进行双向索引
- 可以进行如下操作：+，*，切片
- 可以进行成员判断：in，not in
- 可以进行比较运算



内容

- Python中变量的存储
- 列表类型
- 元组类型
- 集合类型
- 字典类型



基本序列类型：列表（List）

列表是有序、可变的组合数据类型

[<item1>, <item2>...]

- 列表中的元素类型可以不同
- 可以对列表元素进行增删
- 可以对列表元素进行修改



基本序列类型：列表 (List)

列表 (Python)

VS

数组 (C)

- 元素类型可以不同
- 长度可以改变

- 元素类型必须相同
- 长度不可改变



列表的基本操作：列表创建

- 直接赋值

```
>>> list1 = ['physics', 'chemistry', 1997, 2022]
>>> list1
['physics', 'chemistry', 1997, 2022]
```

```
>>> list2 = []
>>> list2
[]
```

- 通过list()函数将其他类型的数据转换为列表

```
>>> list3 = list((1, 3, 5, 7))
>>> list3
[1, 3, 5, 7]
```

```
>>> list4 = list(range(2, 8, 2))
>>> list4
[2, 4, 6]
```

```
>>> list5 = list('Python')
>>> list5
['P', 'y', 't', 'h', 'o', 'n']
```



列表的基本操作：列表元素的增加

- append()方法

`s.append(x)`

- extend()方法

`s.extend(t)`

```
>>> list4.extend([True, None])
>>> list4
[2, 4, 6, -1, 3, 5, 'a', 'b', True, None]
```

```
>>> list4 = [2, 4, 6]
>>> list4.append(-1)
>>> list4
[2, 4, 6, -1]
>>> list4.extend((3, 5))
>>> list4
[2, 4, 6, -1, 3, 5]
>>> list4.extend('ab')
>>> list4
[2, 4, 6, -1, 3, 5, 'a', 'b']
```



列表的基本操作：列表元素的增加

- insert()方法

`s.insert(i, x)`

```
>>> list5  
[1, 2, 3]
```

```
>>> list5.insert(2, 0)
```

```
>>> list5  
[1, 2, 0, 3]
```

```
>>> list5.insert(-3, -5)
```

```
>>> list5  
[1, -5, 2, 0, 3]
```

```
>>> list5.insert(10, 7)
```

```
>>> list5  
[1, -5, 2, 0, 3, 7]
```

```
>>> list5.insert(-12, 8)
```

```
>>> list5  
[8, 1, -5, 2, 0, 3, 7]
```




列表的基本操作：列表元素的增加

- 通过+在列表中增加元素

$s = s + [x, y...]$

```
>>> list3 = [1, 3, 5]
>>> list3 = list3 + [7, 9]
>>> list3
[1, 3, 5, 7, 9]
```

- 通过*在列表中增加重复元素

$s = s * n$

```
>>> list4 = [2, 5, 8]
>>> list4 = list4 * 3
>>> list4
[2, 5, 8, 2, 5, 8, 2, 5, 8]
```



列表的基本操作：索引和切片

- 基本索引用于列表元素的查找、修改与删除

`s[i]`

`s[i] = value`

`del s[i]`

```
>>> list3[8]
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    list3[8]
IndexError: list index out of range
```

```
>>> list3
[1, 3, 5, 7, 9]
>>> x, y = list3[2], list3[-4]
>>> print(x, y)
5 3
>>> list3[1], list3[-2] = 0, 8
>>> list3
[1, 0, 5, 8, 9]
>>> del list3[3]
>>> list3
[1, 0, 5, 9]
```



列表的基本操作：索引和切片

- 切片用于多个列表元素的查找、修改与删除

`s[start: stop: step]`

`s[start: stop: step] = [v1, v2, ...]`

```
>>> list3[1: 3] = [1, 2, 3]
```

```
>>> list3  
[-1, 1, 2, 3, 9]
```

```
>>> list3[1: 3] = [5]
```

```
>>> list3  
[-1, 5, 3, 9]
```

```
>>> list3  
[1, 0, 5, 9]
```

```
>>> x, y = list3[0: 3: 2], list3[-1: -4: -1]
```

```
>>> x  
[1, 5]
```

```
>>> y  
[9, 5, 0]
```

```
>>> list3[0: 3: 2] = [-1, -6]
```

```
>>> list3  
[-1, 0, -6, 9]
```



列表的基本操作：索引和切片

- 切片用于多个列表元素的查找、修改与删除

`s[start: stop: step]`

`s[start: stop: step] = [v1, v2, ...]`

```
>>> list3
[-1, 5, 3, 9]
>>> list3[3: 0: -1] = [1, 2, 3]
>>> list3
[-1, 3, 2, 1]
```

```
>>> list3[3: 0: -1] = [4, 5, 6, 7]
Traceback (most recent call last):
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    list3[3: 0: -1] = [4, 5]
ValueError: attempt to assign sequence of size 2 to extended slice of size 3
```



列表的基本操作：索引和切片

- 切片用于多个列表元素的查找、修改与删除

`s[start: stop: step]`

`s[start: stop: step] = [v1, v2, ...]`

`del s[start: stop: step]`

```
>>> list3 = [1, 3, 5, 7, 9]
```

```
>>> del list3[:4:2]
```

```
>>> list3  
[3, 7, 9]
```

```
>>> del list3[4: 5]
```

```
>>> list3  
[3, 7, 9]
```



列表元素的其它查询方法

- 使用index方法查询值出现的位置

`s.index(x, start = 0, stop = l)`

```
>>> list4 = [1, 2, 1, 3, 5, 1, 7, 9, 11]
>>> list4.index(1)
0
>>> list4.index(1, 2)
2
>>> list4.index(1, 4, 8)
5
>>> list4.index(1, 6)
Traceback (most recent call last):
  File "<pyshell#118>", line 1, in <module>
    list4.index(1, 6)
ValueError: 1 is not in list
```



列表元素的其它删除方法

- 使用pop方法删除默认或指定元素

`s.pop(index = -1)`

```
>>> list5 = [1, 2, 3, 4, 5]
>>> x = list5.pop()
>>> x
5
>>> list5
[1, 2, 3, 4]
```

```
>>> y = list5.pop()
>>> y
3
>>> list5
[1, 2, 4]
>>> z = list5.pop()
>>> z
2
>>> list5
[1, 4]
```

```
>>> list5.pop(3)
Traceback (most recent call last):
  File "<pyshell#131>", line 1, in <module>
    list5.pop(3)
IndexError: pop index out of range
```

```
>>> list5 = []
>>> list5.pop()
Traceback (most recent call last):
  File "<pyshell#133>", line 1, in <module>
    list5.pop()
IndexError: pop from empty list
```



列表元素的其它删除方法

- 使用remove方法删除指定元素

`s.remove(value)`

```
>>> list6 = [1, 3, 5, 1, 7, 9]
>>> list6.remove(1)
>>> list6
[3, 5, 1, 7, 9]
```

```
>>> list6.remove(2)
Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    list6.remove(2)
ValueError: list.remove(x): x not in list
```




列表元素的其它删除方法

- 使用clear方法删除全部元素

`s.clear()`

```
>>> list6
[3, 5, 1, 7, 9]
>>> list6.clear()
>>> list6
[]
```

- 使用del list[:] 删除全部元素

```
>>> list7 = [1, 2, 3]
>>> del list7[:]
>>> list7
[]
```



列表的基本操作：列表删除

- 使用del删除整个列表

del s

```
>>> list8 = [1, 2, 3]
>>> del list8
>>> list8
Traceback (most recent call last):
  File "<pyshell#149>", line 1, in <module>
    list8
NameError: name 'list8' is not defined. Did you mean: 'list5'?
```



列表的成员判断操作

- 使用in判断元素是否在列表中

`x in s`

```
>>> list9 = [1, 'a', -5]
```

```
>>> 'a' in list9
```

```
True
```

```
>>> 5 in list9
```

```
False
```

- 使用 not in判断元素是否不在列表中

`x not in s`

```
>>> 1 not in list9
```

```
False
```

```
>>> 'A' not in list9
```

```
True
```



列表的比较操作

列表之间可以通过比较运算符“比大小”

- 只有同一位置类型相同的列表才能进行比较
- 从第一个元素顺序开始比较，如果相等，则继续，返回第一个不相等元素比较的结果
- 如果所有元素比较均相等，则长的列表大，一样长则两列表相等

```
>>> s1 = [3, 'abc', 4]
>>> s2 = [3, 'abcc', 2]
>>> s3 = [3, 'abc', 2]
>>> s4 = [3, 'abc', 2, 'c']
>>> print( s1 != s2, s1 > s2, s2 > s3, s3 > s4)
True False True False
```

s1、s2、s3和s4的大小关系？

s3 < s4 < s1 < s2



列表的统计操作

- 通过len函数求列表的长度

`l = len(s)`

- 通过max、min函数求列表元素中的最大值、最小值

`ma = max(s)`

`mi = min(s)`

```
>>> list1 = ['abc', 3, -2.5, True]
>>> len(list1)
4
>>> list2 = []
>>> len(list2)
0
>>> list3 = [0, 8, -2, 3, -5, 1.8]
>>> print(max(list3), min(list3))
8 -5
>>> list4 = ['c', 'f', 'E', 'H']
>>> print(max(list4), min(list4))
f E
>>> list5 = ['a', 'AB', 'c1']
>>> print(max(list5), min(list5))
c1 AB
```



列表的统计操作

- 通过count()方法求值出现的次数

`n = s.count(value)`

```
>>> list6 = [1, 0, 'a', 1, -2, False, -1, 1, 1, 0]
```

```
>>> list6.count(1)
```

```
4
```

```
>>> list6.count('a')
```

```
1
```

```
>>> list6.count(2)
```

```
0
```

```
>>> list6.count(True)
```

```
4
```

```
>>> list6.count(False)
```

```
3
```



列表的布尔操作

- 通过all()函数判断列表中元素的布尔值是否全为真

all(s)

```
>>> list1 = [1, 'a', -1, True]
>>> all(list1)
True
```

```
>>> list2 = ['', 1]
>>> all(list2)
False
```

```
>>> list3 = [1, 0]
>>> all(list3)
False
```

```
>>> list4 = []
>>> all(list4)
True
```



列表的布尔操作

- 通过any()函数判断列表中元素的布尔值是否有真

any(s)

```
>>> list5 = [0, "", False, []]  
>>> any(list5)  
False
```

```
>>> list7 = []  
>>> any(list7)  
False
```

```
>>> list6 = [0, 1]  
>>> any(list6)  
True
```




列表的布尔操作

```
>>> list7 = []  
>>> any(list7)  
False  
>>> all(list7)  
True
```

```
def all(iterable):  
    for element in iterable:  
        if not element:  
            return False  
    return True
```

<https://docs.python.org/3/library/functions.html#all>

```
def any(iterable):  
    for element in iterable:  
        if element:  
            return True  
    return False
```

<https://docs.python.org/3/library/functions.html#any>



列表的排序操作

- 通过sort()方法对列表元素进行排序，并修改列表

`s.sort(key = None, reverse = False)`

```
>>> list1 = [0, 8, -2, 3, -5]
>>> list1.sort()
>>> list1
[-5, -2, 0, 3, 8]
```

```
>>> list2 = [0, -2, -3, True, False]
>>> list2.sort(reverse = True)
>>> list2
[True, 0, False, -2, -3]
```



列表的排序操作

- 通过参数key修改排序依据

`s.sort(key = func, reverse = False)`

```
def k1(x):  
    return (x * x - 4*x + 4)
```

```
>>> list3 = [1, 5, -3, 2, 8]  
>>> list3.sort(key = k1)  
>>> list3  
[2, 1, 5, -3, 8]
```

```
>>> for i in list3:  
...     print('i =', i, 'k1(i) = ', k1(i))  
  
...  
i = 1 k1(i) = 1  
i = 5 k1(i) = 9  
i = -3 k1(i) = 25  
i = 2 k1(i) = 0  
i = 8 k1(i) = 36
```



列表的排序操作

- 通过sorted()函数对列表元素进行排序，返回新列表

`sorted(s, key = None, reverse = False)`

```
>>> list1 = [0, 8, -2, 3, -5]
>>> sorted(list1)
[-5, -2, 0, 3, 8]
>>> list1
[0, 8, -2, 3, -5]
>>> sorted(list1, key = k1, reverse = True)
[-5, 8, -2, 0, 3]
```

```
>>> for i in list1:
...     print('i =', i, 'k1(i) = ', k1(i))
...
i = 0 k1(i) = 4
i = 8 k1(i) = 36
i = -2 k1(i) = 16
i = 3 k1(i) = 1
i = -5 k1(i) = 49
```



列表的反序操作

- 通过reverse()方法对列表元素进行反序操作，并修改列表
`s.reverse()`

```
>>> list4 = ['a', 3, -5, 'cd', True]
>>> list4.reverse()
>>> list4
[True, 'cd', -5, 3, 'a']
```

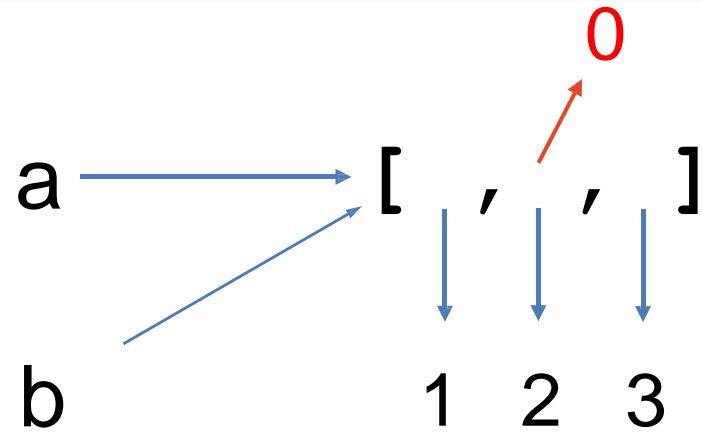


列表的复制

- 直接赋值

s2 = s1

```
>>> a = [1, 2, 3]
>>> b = a
>>> b
[1, 2, 3]
>>> a[1] = 0
>>> a
[1, 0, 3]
>>> b
[1, 0, 3]
```



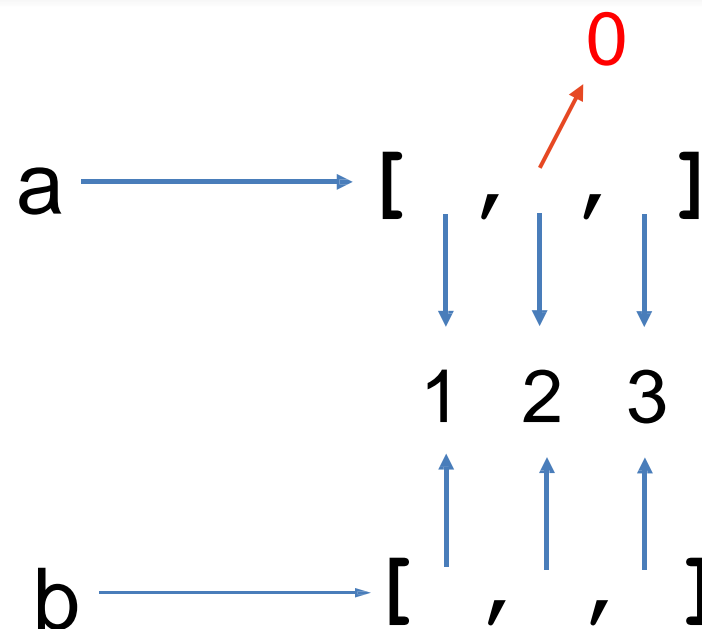


列表的复制

```
>>> a[1] = 0
```

- 浅复制 `s2 = s1.copy()`

```
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> b
[1, 2, 3]
>>> a[1] = 0
>>> a
[1, 0, 3]
>>> b
[1, 2, 3]
```

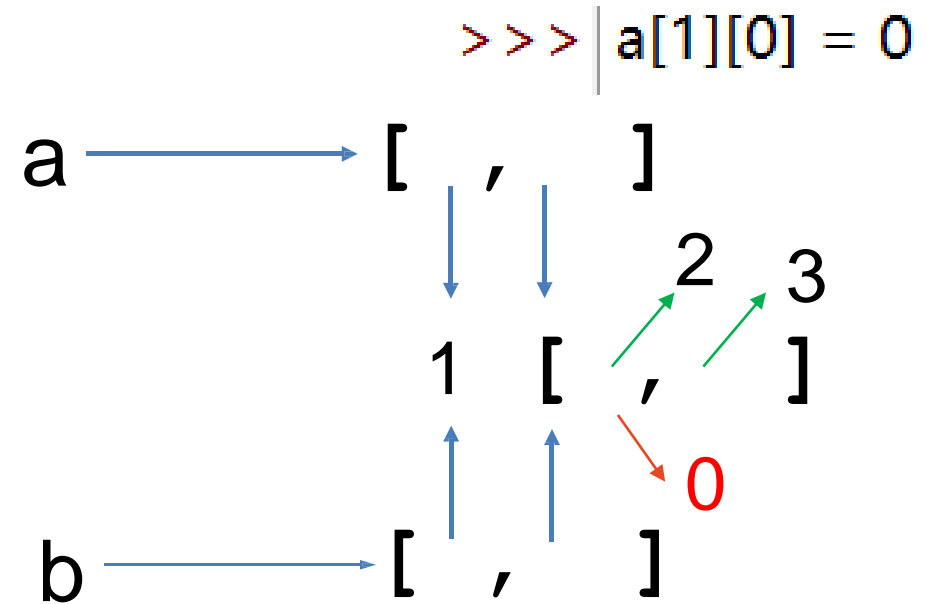




列表的复制

- 浅复制 `s2 = s1.copy()`

```
>>> a = [1, [2, 3]]
>>> b = a.copy()
>>> b
[1, [2, 3]]
>>> a[1][0] = 0
>>> a
[1, [0, 3]]
>>> b
[1, [0, 3]]
```





多重列表乘法操作存在的问题

```
>>> x = [[1, 2], 3]
>>> y = x * 3
>>> y
[[1, 2], 3, [1, 2], 3, [1, 2], 3]
>>> y[2][0] = 5
>>> y
[[5, 2], 3, [5, 2], 3, [5, 2], 3]
>>> x
[[5, 2], 3]
```

#四个元素[1,2]是对同一个子列表的引用

#通过修改子列表可以同时修改四个元素的值（修改x[0][0]效果相同）

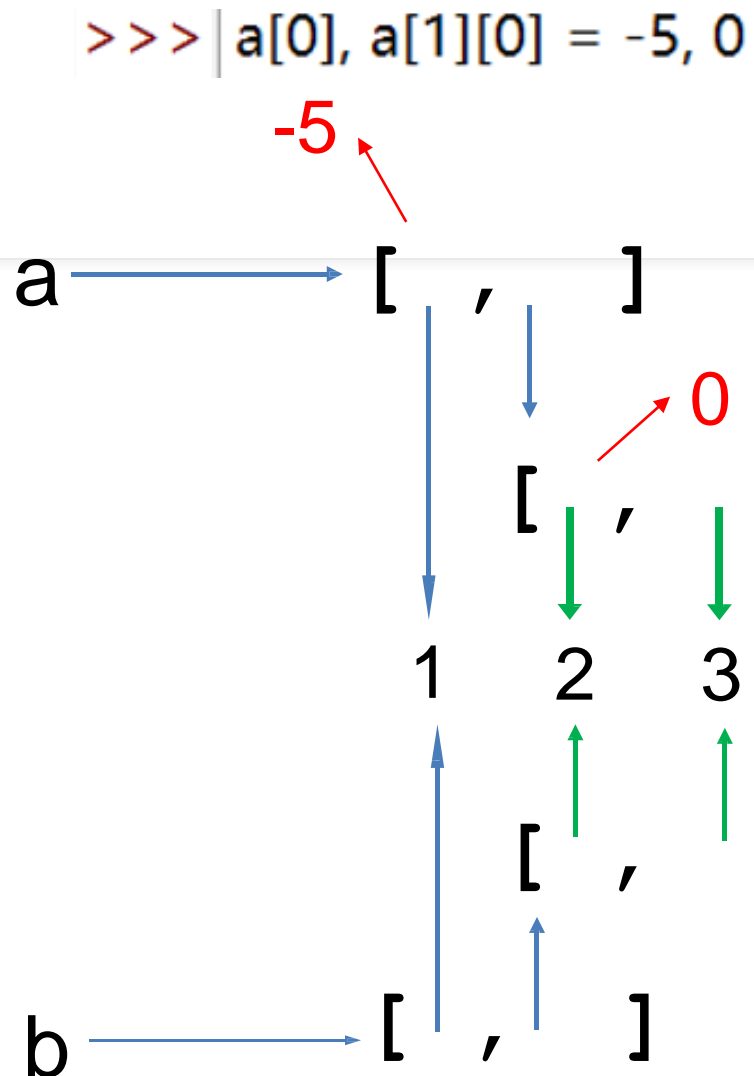


列表的复制

- 深复制 *import copy*

s2 = copy.deepcopy(s1)

```
>>> import copy
>>> a = [1, [2, 3]]
>>> b = copy.deepcopy(a)
>>> b
[1, [2, 3]]
>>> a[0], a[1][0] = -5, 0
>>> a
[-5, [0, 3]]
>>> b
[1, [2, 3]]
```





列表常用方法总结

函数或方法	描述
s.append(x)	将x添加到列表s末尾，列表长度加1
s.insert(i, x)	在列表s中序号为i的位置插入对象x，i以后的元素后移
s.extend(t)	把可迭代对象t的成员附加到列表s的尾部
s.copy()	复制列表s为一个新列表，返回给调用者，复制过程为浅拷贝
del s[i]	删除列表s中序号为i的元素，i以后的元素前移
s.pop(i)	删除列表s末尾或序号为i的位置的元素，返回该元素的值
s.remove(x)	删除列表s中第一次出现的x值
s.clear()	清空列表s的元素
del s	删除列表s
s.sort()	对列表s排序，修改列表
sorted(s)	对列表s排序，生成新列表
s.reverse()	将列表s反序，修改列表



列表解析（列表推导式）

通过对一个或多个列表中的每个元素应用某种操作，从而快速将一个或多个列表映射为另一个列表

[exp for v1 in s1 (for v2 in s2)(if exp1 if exp2)... ...]

```
>>> x = [5, 8, 3]
>>> y = [2, 9, 6, 7]
>>> s = [a*b for a in x for b in y if a < b]
>>> s
[45, 30, 35, 72, 27, 18, 21]
```



列表解析（列表推导式）

```
>>> list1 = [x**2 for x in range(5)]
>>> list1
[0, 1, 4, 9, 16]
>>> list2 = [x for x in range(30) if x % 3 == 0]
>>> list2
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
>>> list3 = [i**2 if i < 5 else i * 2 for i in range(10)]
>>> list3
[0, 1, 4, 9, 16, 10, 12, 14, 16, 18]
```



列表解析（列表推导式）

```
>>> x, y = [1, 3, 5], ['b', 'd', 'a']
>>> list4 = [str(i) + j for i in x for j in y]
>>> list4
['1b', '1d', '1a', '3b', '3d', '3a', '5b', '5d', '5a']
>>> list5 = [[j*2, i] for i in x for j in y if i > 1]
>>> list5
[['bb', 3], ['dd', 3], ['aa', 3], ['bb', 5], ['dd', 5], ['aa', 5]]
>>> mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> list6 = [[row[i] for row in mat] for i in (0, 1, 2)]
>>> list6
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```



内容

- Python中变量的存储
- 列表类型
- 元组类型
- 集合类型
- 字典类型



序列类型：元组 (tuple)

元组是有序、不可变的组合数据类型

(<item1>, <item2>...)

- 元组中的元素类型可以不同

```
>>> tup1 = ('physics', 'chemistry', 1997, True)
>>> tup1
('physics', 'chemistry', 1997, True)
```

- 不能对元组元素进行增删改

```
>>> tup1[2] = 5
Traceback (most recent call last):
  File "<pyshell#149>", line 1, in <module>
    tup1[2] = 5
TypeError: 'tuple' object does not support item assignment
```




序列类型：元组 (tuple)

元组

VS

列表

- 元素不能修改
- 长度不能改变

- 元素可以修改
- 长度可以改变



元组的基本操作：元组创建

- 直接赋值

```
>>> tup1 = ('physics', 'chemistry', 1997, True)
>>> tup1
('physics', 'chemistry', 1997, True)

>>> tup2 = 1, 2, 3, 'abc'
>>> tup2
(1, 2, 3, 'abc')

>>> tup3 = ()
>>> tup3
()
```

```
>>> tup4 = (1)
>>> tup4
1
```

```
>>> tup4 = (1, )
>>> tup4
(1,)
```

```
>>> tup4 = 1,
>>> tup4
(1,)
```



元组的基本操作：元组创建

- 通过tuple()函数将其他类型的数据转换为元组

```
>>> tup5 = tuple(['a', False, 1.7, -2])
>>> tup5
('a', False, 1.7, -2)
```

```
>>> tup7 = tuple(range(2, 8, 2))
>>> tup7
(2, 4, 6)
```

```
>>> tup6 = tuple('Python')
>>> tup6
('P', 'y', 't', 'h', 'o', 'n')
```

```
>>> tup8 = tuple(x**2 if x < 5 else x*2 for x in range(10))
>>> tup8
(0, 1, 4, 9, 16, 10, 12, 14, 16, 18)
```



元组的基本操作：索引和切片

- 基本索引用于元组元素的查找

`t[i]`

```
>>> tup1 = (1, 3, 5, 7, 9)
>>> x, y = tup1[1], tup1[-2]
>>> print(x, y)
3 7
```

- 不能通过索引进行元组元素的修改

```
>>> tup1[2] = -1
Traceback (most recent call last):
  File "<pyshell#181>", line 1, in <module>
    tup1[2] = -1
TypeError: 'tuple' object does not support item assignment
```



元组的基本操作：索引和切片

- 不能通过索引进行元组元素的删除

```
>>> tup1
(1, 3, 5, 7, 9)
>>> del tup1[3]
Traceback (most recent call last):
  File "<pyshell#184>", line 1, in <module>
    del tup1[3]
TypeError: 'tuple' object doesn't support item deletion
```



元组的基本操作：索引和切片

- 切片用于多个元组元素的查找

`t[start: stop: step]`

```
>>> tup1
      (1, 3, 5, 7, 9)
>>> x, y = tup1[: 3: 2], tup1[-1: -4: -1]
>>> x
      (1, 5)
>>> y
      (9, 7, 5)
```



元组的基本操作：索引和切片

- 不能通过切片进行多个元组元素的修改和删除

```
>>> tup1
(1, 3, 5, 7, 9)
>>> tup1[0: 2] = (2, 4)
Traceback (most recent call last):
  File "<pyshell#199>", line 1, in <module>
    tup1[0: 2] = (2, 4)
TypeError: 'tuple' object does not support item assignment

>>> del tup1[0: 2]
Traceback (most recent call last):
  File "<pyshell#200>", line 1, in <module>
    del tup1[0: 2]
TypeError: 'tuple' object does not support item deletion
```



元组元素的其它查询方法

- 使用index方法查询值出现的位置

`t.index(x, start = 0, stop = l)`

```
>>> tup2 = (1, -2, 1, 3, 5, 1, 0, 9, 7)
>>> tup2.index(1)
0
```

```
>>> tup2.index(1, 2)
2
```

```
>>> tup2.index(1, 4, 8)
5
```

```
>>> tup2.index(1, 6)
Traceback (most recent call last):
  File "<pyshell#205>", line 1, in <module>
    tup2.index(1, 6)
ValueError: tuple.index(x): x not in tuple
```




元组的基本操作：元组删除

- 使用del删除整个元组

del t

```
>>> tup1
(1, 3, 5, 7, 9)
>>> del tup1
>>> tup1
Traceback (most recent call last):
  File "<pyshell#209>", line 1, in <module>
    tup1
NameError: name 'tup1' is not defined. Did you mean: 'tup2'?
```



元组的成员判断操作

- 使用in判断元素是否在元组中

`x in t`

- 使用 not in判断元素是否不在元组中

`x not in t`

```
>>> tup3 = (1, 'a', 5)
```

```
>>> 'A' in tup3
```

```
False
```

```
>>> True in tup3
```

```
True
```

```
>>> tup3 = (1, 'a', 5)
```

```
>>> 'a' not in tup3
```

```
False
```

```
>>> 8 not in tup3
```

```
True
```



元组的比较操作

元组之间可以通过比较运算符“比大小”>>>

- 只有同一位置类型相同的元组才能进行比较
- 从第一个元素顺序开始比较，如果相等，则继续，返回第一个不相等元素比较的结果
- 如果所有元素比较均相等，则长的元组大，一样长则两元组相等

```
>>> (1, 2) < (1, 'a')
```

```
Traceback (most recent call last):  
File "<pyshell#217>", line 1, in <module>  
(1, 2) < (1, 'a')
```

```
TypeError: '<' not supported between instances of 'int' and 'str'
```

```
>>> t1 = (3, 'abc', 4)
```

```
>>> t2 = (3, 'abc', 2)
```

```
>>> t3 = (3, 'abcc', 1)
```

```
>>> t4 = (3, 'abc', 2, 'c')
```

```
>>> print(t1 < t2, t1 < t3, t1 < t4)  
False True False
```

```
>>> print(t2 < t3, t2 < t4)  
True True
```

```
>>> print(t3 < t4)  
False
```



元组的统计操作

函数或方法	描述
len(t)	返回元组中元素的个数
max(t)	返回元组元素中的最大值，如元组中元素数据类型不同，或属于无法比较大小的数据类型，则报错
min(t)	返回元组元素中的最小值，如元组中元素数据类型不同，或属于无法比较大小的数据类型，则报错
t.count(x)	返回元组元素值出现x的次数



元组的布尔操作

- 通过all()函数判断元组中的值是否全为真

all(t)

- 通过any()函数判断元组中是否有值为真

any(t)

```
>>> tup4 = (1, 'a', -1, True)
>>> all(tup4)
True
>>> any(tup4)
True
```

```
>>> tup5 = ('', ' ')
>>> all(tup5)
False
>>> any(tup5)
True
```

```
>>> tup6 = ()
>>> all(tup6)
True
>>> any(tup6)
False
```



元组的排序操作

- 通过sorted()函数对元组元素进行排序，返回一个列表

`sorted(t, key = None, reverse = False)`

```
>>> tup1 = (2, 5, -3, 0)
```

```
>>> sorted(tup1)
```

```
[-3, 0, 2, 5]
```

```
>>> tup1 = (2, 5, -3, 0)
```

```
>>> tup2 = tuple(sorted(tup1))
```

```
>>> tup2
```

```
(-3, 0, 2, 5)
```

```
>>> sorted(tup1, key = k1, reverse = True)
```

```
[-3, 5, 0, 2]
```

```
>>> def k1(i):
```

```
...     return i ** 2 - 4 * i + 4
```

```
...
```

```
i = 2 k1(i) = 0
```

```
i = 5 k1(i) = 9
```

```
i = -3 k1(i) = 25
```

```
i = 0 k1(i) = 4
```



元组的“修改”操作——“假修改”

通过非原地操作的方法，实现元组的修改

- 通过+在元组中增加元素

$t = t + (x, y...)$

- 通过*在元组中增加重复元素

$t = t * n$

- 通过切片或切片和+的组合在元组中删除元素

$t = t[start1: stop1: step1] + t[start2: stop2: step2]$

```
>>> tup4 = (1, 3, 5)
>>> tup4 = tup4 + (7, 9)
>>> tup4
(1, 3, 5, 7, 9)
>>> tup4 = tup4 * 2
>>> tup4
(1, 3, 5, 7, 9, 1, 3, 5, 7, 9)
```

```
>>> tup4 = tup4[: 3] + tup4[5:]
>>> tup4
(1, 3, 5, 1, 3, 5, 7, 9)
```



元组的“修改”操作——“真修改”

当元组中存在可变数据类型的元素时，可以对该元素的值进行修改

```
>>> a = (1, [2, 3])
>>> id(a)
2036700180352
>>> a[1][0] = -5
>>> a
(1, [-5, 3])
>>> id(a)
2036700180352
```

```
>>> a1 = (1, 2, 3)
>>> a2 = (1, 2, 3)
>>> a1 is a2
False
>>> a1 == a2
True
```




元组的复制操作

- 赋值

```
tuple2 = tuple1
```

- 浅复制

```
tuple2 = copy.copy(tuple1)
```

- 深复制

```
tuple2 = copy.deepcopy(tuple1)
```



列表常用方法总结

函数或方法	描述
<code>s.append(x)</code>	将x添加到列表s末尾，列表长度加1
<code>s.insert(i, x)</code>	在列表s中序号为i的位置插入对象x，i以后的元素后移
<code>s.extend(t)</code>	把可迭代对象t的成员附加到列表s的尾部
<code>s.copy()</code>	复制列表s为一个新列表，返回给调用者，复制过程为浅拷贝
<code>del s[i]</code>	删除列表s中序号为i的元素，i以后的元素前移
<code>s.pop(i)</code>	删除列表s末尾或序号为i的位置的元素，返回该元素的值
<code>s.remove(x)</code>	删除列表s中第一次出现的x值
<code>s.clear()</code>	清空列表s的元素
<code>del s</code>	删除列表s
<code>s.sort()</code>	对列表s排序，修改列表
<code>sorted(s)</code>	对列表s排序，生成新列表
<code>s.reverse()</code>	将列表s反序，修改列表



使用元组的好处

- 元组的操作速度比列表快。如果需要使用一个常量集进行遍历操作，则应使用元组而非列表
- 使用元组相当于进行“写保护”，可以使代码更安全
- 元组和列表间的相互转换，可以认为是对一组数据的“加锁”和“解锁”



内容

- Python中变量的存储
- 列表类型
- 元组类型
- 集合类型
- 字典类型



无序类型：集合 (set)

集合是无序、可变的组合数据类型，集合的概念和数学上的集合相同

`{<item1>, <item2> ...}`

- 集合中的元素类型可以不同
- 集合中不能有重复元素
- 可以对集合元素进行增删
- 集合中的元素必须是不可变数据类型（可哈希）
- 如果元组中包含可变数据类型的元素，则该元组不能作为集合中的元素



集合的基本操作：集合创建

```
>>> s7 = set(x * y for x in range(5) for y in range(4) if x < y)
>>> s7
{0, 2, 3, 6}
```

- 直接赋值

```
>>> s1 = {1, 2, 'ok', 1, (0, 6)}
>>> s1
{1, 2, 'ok', (0, 6)}
```

```
>>> s2 = set()
>>> s2
set()
```

- 通过set()函数将其他类型的数据转换为集合

```
>>> s3 = set((1, 2, 'ok'))
>>> s3
{1, 2, 'ok'}
```

```
>>> s4 = set([1, 3, 3, 5, 7, 1])
>>> s4
{1, 3, 5, 7}
```

```
>>> s5 = set('Hello')
>>> s5
{'o', 'e', 'H', 'l'}
```

```
>>> s6 = set(range(1, 7, 2))
>>> s6
{1, 3, 5}
```



集合的基本操作：集合元素的增加

- add()方法

s.add(x)

```
>>> s6
{1, 3, 5}
>>> s6.add(-1)
>>> s6
{1, 3, 5, -1}
>>> s6.add(3)
>>> s6
{1, 3, 5, -1}
```

```
>>> s6.add('abc')
>>> s6
{1, 3, 'abc', 5, -1}
>>> s6.add(('A', 8))
>>> s6
{1, 3, 'abc', 5, ('A', 8), -1}
>>> s6.add([0, 7])
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    s6.add([0, 7])
TypeError: unhashable type: 'list'
```



集合的其它增加元素的方法

- update()方法

s.update(t)

```
>>> s4
{1, 3, 5, 7}
>>> s4.update({0, -2, 3, 9})
>>> s4
{0, 1, 3, 5, 7, 9, -2}
>>> s4.update([2, 1])
>>> s4
{0, 1, 2, 3, 5, 7, 9, -2}
```

```
>>> s4.update('abc')
>>> s4
{0, 1, 2, 3, 'b', 5, 7, 9, 'a', 'c', -2}
>>> s4.update((1, [2, 3]))
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    s4.update((1, [2, 3]))
TypeError: unhashable type: 'list'
```




集合的基本操作：集合元素的删除

- 使用remove方法删除指定元素

`s.remove(x)`

```
>>> s6
{1, 3, 'abc', 5, ('A', 8), -1}
>>> s6.remove(5)
>>> s6
{1, 3, 'abc', ('A', 8), -1}
```

```
>>> s6.remove(5)
Traceback (most recent call last):
  File "<pyshell#63>", line 1, in <module>
    s6.remove(5)
KeyError: 5
```



集合的基本操作：集合元素的删除

- 使用discard()方法删除指定元素

s.discard(x)

```
>>> s7 = set('abcd')
>>> s7
{'d', 'c', 'a', 'b'}
>>> s7.discard('b')
>>> s7
{'d', 'c', 'a'}
```

```
>>> s7.discard('A')
>>> s7
{'d', 'c', 'a'}
```



集合的基本操作：集合元素的删除

- 使用pop()方法随机删除一个元素，并返回被删除元素的值

s.pop()

```
>>> s7
{'d', 'c', 'a'}
>>> s7.pop()
'd'
>>> s7
{'c', 'a'}
>>> s7.pop()
'c'
>>> s7
{'a'}
```

```
>>> s7.pop()
'a'
>>> s7
set()
>>> s7.pop()
Traceback (most recent call last):
  File "<pyshell#77>", line 1, in <module>
    s7.pop()
KeyError: 'pop from an empty set'
```



集合的基本操作：集合的清空和删除

- 使用clear()方法清空集合中的元素

`s.clear()`

```
>>> s1
{1, 2, 'ok', (0, 6)}
>>> s1.clear()
>>> s1
set()
```

- 使用del删除集合

`del s`

```
>>> s3
{1, 2, 'ok'}
>>> del s3
>>> s3
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    s3
NameError: name 's3' is not defined. Did you mean: 's1'?
```



集合的成员判断操作

- 使用in判断元素是否在集合中

`x in s`

- 使用 not in判断元素是否不在集合中

`x not in s`

```
>>> s5
{'o', 'e', 'H', 'l'}
>>> "e" in s5
True
>>> "h" in s5
False
>>> "ll" not in s5
True
```

集合用于需要经常判断一个东西是否在一堆东西里的场合，其时间复杂度为 $O(1)$



集合的统计操作和布尔操作

函数或方法	描述
len(s)	返回集合s中元素的个数
max(s)	返回集合s中元素的最大值，如集合中元素数据类型不同，或属于无法比较大小的数据类型，则报错
min(s)	返回集合s中元素的最小值，如集合中元素数据类型不同，或属于无法比较大小的数据类型，则报错
all(s)	判断集合s中元素是否全为True
any(s)	判断集合s中是否有元素为True



集合的排序操作

- 通过sorted()函数对集合元素进行排序，返回一个列表
`sorted(s, key = None, reverse = False)`

```
>>> set1 = {2, 5, -3, 0}
>>> sorted(set1)
[-3, 0, 2, 5]
>>> set1 = {2, 5, -3, 0}
>>> set2 = set(sorted(set1))
>>> set2
{0, 2, 5, -3}
```

```
>>> sorted(set1, key = k1, reverse = True)
[-3, 5, 0, 2]
```

```
>>> def k1(i):
...     return i ** 2 - 4 * i + 4
```

```
...
i = 2 k1(i) = 0
i = 5 k1(i) = 9
i = -3 k1(i) = 25
i = 0 k1(i) = 4
```



集合的复制操作

- 赋值

```
set2 = set1
```

- 浅复制

```
set2 = set1.copy()
```

```
set2 = copy.copy(set1)
```

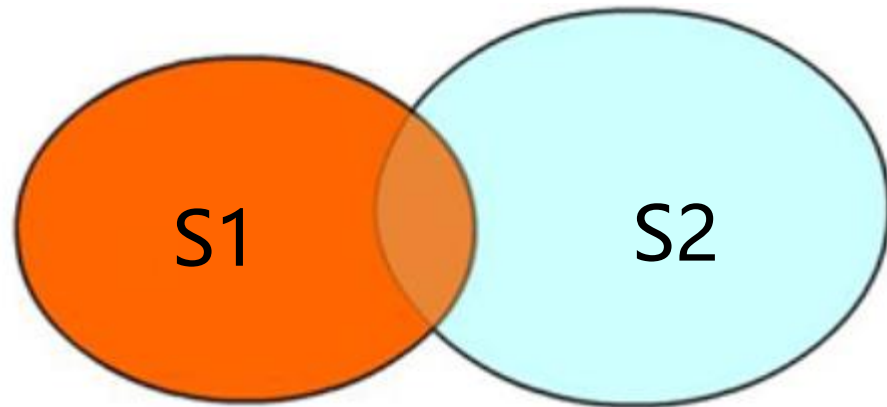
- 深复制

```
set2 = copy.deepcopy(set1)
```




集合运算

表达式	描述
$s1 s2$	返回集合s1和集合s2的并集
$s1\&s2$	返回集合s1和集合s2的交集
$s1-s2$	返回集合s1和集合s2的差集，即由在s1中且不在s2中的元素组成的集合
$s1^{\wedge}s2$	返回集合s1和集合s2的对称差，等价于 $(s1 s2)-(s1\&s2)$





集合比较运算

表达式	描述
<code>s1 == s2</code>	若s1和s2长度相等，元素完全相同，则返回True;否则返回False
<code>s1 != s2</code>	若s1和s2元素不完全相同，则返回True;否则返回False
<code>s1 > s2</code>	若s1是s2的纯超集，则返回True;否则返回False
<code>s1 < s2</code>	若s1是s2的纯子集，则返回True;否则返回False
<code>s1 >= s2</code>	若s1是s2的超集，则返回True;否则返回False
<code>s1 <= s2</code>	若s1是s2的子集，则返回True;否则返回False



集合对象方法实现集合运算

方法	描述
<code>s1.isdisjoint(s2)</code>	如果s1和s2之间没有共同元素，返回True，否则False
<code>s1.issubset(s2)</code>	如果s1是s2的子集，返回True，否则False
<code>s1.isuperset(s2)</code>	如果s1是s2的超集，返回True，否则False
<code>s1.union(s2,...)</code>	返回s1和s2、...的并集
<code>s1.intersection(s2,...)</code>	返回s1和s2、...的交集
<code>s1.difference(s2,...)</code>	返回s1-s2的差集
<code>s1.symmetric_difference(s2)</code>	返回s1和s2的对称差



不可变集合 (frozen set)

不可变集合是无序、不可变的组合数据类型，不可变集合和集合的关系类似于元组和列表的关系

- 创建：frozenset()

```
>>> s1 = {1, 2, 3, 'ok', (1, 0)}
>>> fs1 = frozenset(s1)
>>> fs1
frozenset({1, 2, 3, 'ok', (1, 0)})
```

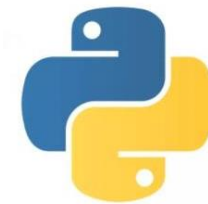
```
>>> fs3 = frozenset('Hello World!')
>>> fs3
frozenset({'W', ' ', 'l', 'H', 'r', 'd', 'e', '!', 'o'})
```

```
>>> fs2 = frozenset([1, 3, 3, 5, 1, 7])
>>> fs2
frozenset({1, 3, 5, 7})
```



不能用于不可变集合的函数或方法

函数或方法	描述
<code>s.add(x)</code>	将x添加到集合s中
<code>s.update(t)</code>	将可迭代对象t中的成员添加到集合s中
<code>s.remove(x)</code>	删除集合s中的元素x, 若x不存在则报错
<code>s.discard(x)</code>	删除集合s中的元素x, x不存在不报错
<code>s.pop()</code>	随机删除集合s中的一个元素, 并返回被删除元素值
<code>s.clear()</code>	清空集合s的元素



内容

- Python中变量的存储
- 列表类型
- 元组类型
- 集合类型
- 字典类型



字典的概念：映射

字典是Python中的一种映射类型

计算机网络	86
计算机组成原理	90
面向对象程序设计实践	88
形式语言与自动机	96
数据结构程序设计	95
Java网络编程	94
形势与政策4	85

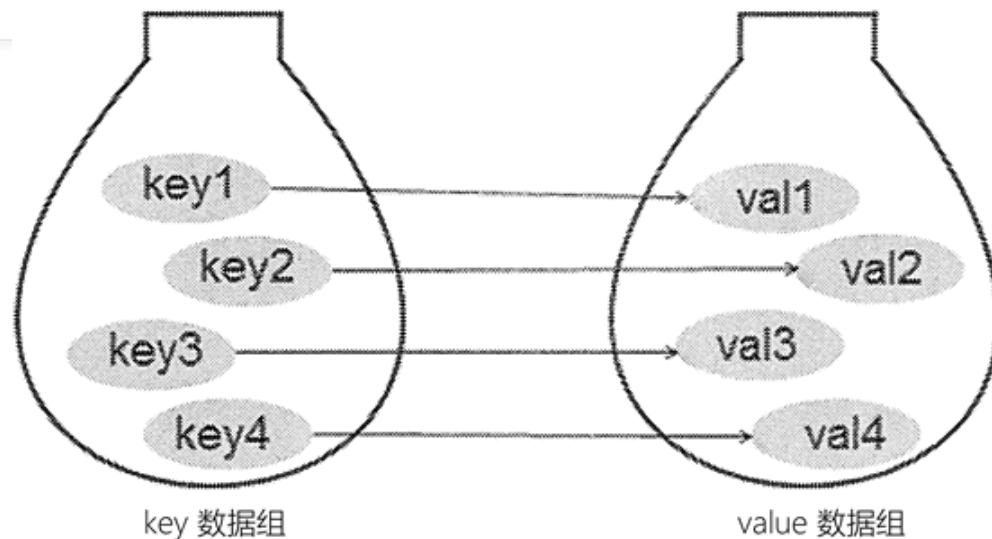
中文姓名	利昂内尔·安德烈斯·梅西·库西蒂尼
外文姓名	Lionel Andrés Messi Cuccitini
国籍	阿根廷
出生地	阿根廷圣菲省罗萨里奥
出生日期	1987年6月24日
身高	1.7m
体重	72kg

字典是Python中表达和存储这种映射关系的数据类型



字典的基本概念

- 字典用于存放具有映射关系 (mapping) 的数据, 是 Python 中唯一内建的映射类型。
- 字典中的每一个元素就是一个 **键值对 (*key-value*)**, 代表 “键” 和 “值” 之间的映射关系 (关联关系)。



`{key1: val1, key2: val2, key3: val3, key4: val4}`



字典的基本特征

字典中的元素通过键进行索引，因此：

- 可以通过键直接访问对应的值
- 字典中的键是唯一的
- 字典中的元素（键值对）是无序的
- 键必须是数字、字符串、元组等不可变数据类型（可哈希）
- 如果元组中包含可变数据类型的元素，则该元组不能作为字典中的键

```
>>> dict1 = {'a': 1, 'b': 2}
>>> dict1['a']
1

>>> dict1 = {'a': 1, 'b': 2}
>>> dict2 = {'b': 2, 'a': 1}
>>> dict1 == dict2
True
```



字典的基本操作：字典创建

- 直接赋值

```
>>> score = {'计算机组成原理': 86, '计算机网络': 90}
>>> score
{'计算机组成原理': 86, '计算机网络': 90}
```

```
>>> dict1 = {}
>>> dict1
{}

```

- 通过dict()函数生成字典

```
>>> items = [('name', 'Mary'), ('age', 21)]
>>> dict2 = dict(items)
>>> dict2
```

```
{'name': 'Mary', 'age': 21} >>> dict3 = dict(name = 'Mary', age = 21, height = 1.67)
>>> dict3
{'name': 'Mary', 'age': 21, 'height': 1.67}
```



字典的基本操作：字典创建

- 通过zip()函数将两个列表或元组组合成字典

```
>>> countries = ['China', 'United States', 'Russia', 'United Kingdom', 'France']
>>> capitals = ('Beijing', 'Washington DC', 'Moscow', 'London', 'Paris')
>>> dict4 = dict(zip(countries, capitals))
>>> dict4
{'China': 'Beijing', 'United States': 'Washington DC', 'Russia': 'Moscow',
 'United Kingdom': 'London', 'France': 'Paris'}
>>> capitals2 = ['Beijing', 'Washington DC', 'Moscow', 'London']
>>> dict5 = dict(zip(countries, capitals2))
>>> dict5
{'China': 'Beijing', 'United States': 'Washington DC', 'Russia': 'Moscow',
 'United Kingdom': 'London'}
```



字典的基本操作：字典创建

- 通过zip()函数将两个列表或元组组合成字典

```
>>> countries = ['China', 'United States', 'Russia', 'United Kingdom', 'France']  
>>> capitals2 = ['Beijing', 'Washington DC', 'Moscow', 'London']
```

```
dict5 = dict(zip(countries, capitals2))
```

```
dict5
```

```
{'China': 'Beijing', 'United States': 'Washington DC', 'Russia': 'Moscow', 'United Kingdom': 'London'}
```

```
dict5 = dict(zip(countries, capitals2, strict=True))
```

```
Traceback (most recent call last):
```

```
File "<pyshell#5>", line 1, in <module>
```

```
    dict5 = dict(zip(countries, capitals2, strict=True))
```

```
ValueError: zip() argument 2 is shorter than argument 1
```



字典的基本操作：字典创建

- 通过fromkeys()方法创建初始字典

```
>>> dict6 = dict.fromkeys(countries)
>>> dict6
{'China': None, 'United States': None, 'Russia': None, 'United Kingdom': None,
'France': None}

>>> dict7 = dict.fromkeys(countries, 'Beijing')
>>> dict7
{'China': 'Beijing', 'United States': 'Beijing', 'Russia': 'Beijing', 'United Kingdom':
'Beijing', 'France': 'Beijing'}
```



字典的基本操作：字典元素的增加及修改

$d[k] = v$

- 在字典中新增一个元素

```
>>> score
{'计算机组成原理': 86, '计算机网络': 90}
>>> score['形式语言与自动机'] = 85
>>> score
{'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 85}
```

- 在字典中修改一个元素

```
>>> score['形式语言与自动机'] = 95
>>> score
{'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 95}
```



字典的基本操作：字典元素的删除

- 在字典中删除一个元素

`del d[k]`

```
>>> score
{'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 95}
>>> del score['计算机网络']
>>> score
{'计算机组成原理': 86, '形式语言与自动机': 95}
```

- 在字典中删除一个元素，并返回被删除的值

`v = d.pop(k)`

```
>>> sc1 = score.pop('形式语言与自动机')
>>> sc1
95
>>> score
{'计算机组成原理': 86}
```



字典元素的删除：避免错误

- 删除不存在的元素，并返回被删除的值

```
>>> score
{'计算机组成原理': 86}
>>> sc2 = score.pop('形式语言与自动机')
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    sc2 = score.pop('形式语言与自动机')
KeyError: '形式语言与自动机'
```

- 删除不存在的元素

```
>>> score
{'计算机组成原理': 86}
>>> del score['计算机网络']
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    del score['计算机网络']
KeyError: '计算机网络'
```




字典元素的删除：避免错误

- 使用pop函数中的default参数

`d.pop(k[, default])`

```
>>> score
{'计算机组成原理': 86}
>>> sc3 = score.pop('形式语言与自动机', -1)
>>> sc3
-1
>>> score
{'计算机组成原理': 86}
```

- 删除前通过操作符in进行判断

```
>>> score
{'计算机组成原理': 86}
>>> '计算机网络' in score
False
>>> '计算机组成原理' in score
True
```



字典的基本操作：字典的清空和删除

- 使用clear()方法清空字典中的元素

d.clear()

```
>>> score
{'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 85}
>>> score.clear()
>>> score
{}
```

- 使用del删除字典

del d

```
>>> score
{'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 85}
>>> del score
>>> score
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    score
NameError: name 'score' is not defined
```



访问字典中的值：避免错误

- 直接访问字典中的值 `d[k]`

```
>>> score = {'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 85}
```

```
>>> score['数据结构']
```

```
Traceback (most recent call last):
```

```
File "<pyshell#48>", line 1, in <module>
```

```
score['数据结构']
```

```
KeyError: '数据结构'
```

- 使用`get()`函数

`d.get(k[, default])`

```
>>> score
```

```
{'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 85}
```

```
>>> sc4 = score.get('计算机网络', -1)
```

```
>>> sc4
```

```
90
```

```
>>> sc5 = score.get('数据结构', -1)
```

```
>>> sc5
```

```
-1
```



字典的遍历

实际应用中的字典往往包含大量键值对。对字典进行遍历，在遍历过程中根据需要对字典元素进行操作，是字典最为常见的应用方式

- 获得字典中所有的键值对、键或值

函数或方法	描述
d.items()	获得字典中所有的键值对
d.keys()	获得字典中所有的键
d.values()	获得字典中所有的值

- 使用for循环进行遍历



字典的遍历

- 获得字典中所有的键值对、键或值

```
>>> score
{'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 85}
>>> score.items()
dict_items([('计算机组成原理', 86), ('计算机网络', 90), ('形式语言与自动机', 85)])
>>> score.keys()
dict_keys(['计算机组成原理', '计算机网络', '形式语言与自动机'])
>>> score.values()
dict_values([86, 90, 85])
```



字典遍历的应用

- 使用 for ... in 实现对字典的遍历

File Edit Format Run Options Window Help

```
#dictExample1.py
```

```
score = {'计算机组成原理': 86, '计算机网络': 90, '形式语言与自动机': 85}
```

```
for key, value in score.items():
```

```
    print('我的{}成绩是: {}'.format(key, value))
```

```
我的计算机组成原理成绩是: 86
```

```
我的计算机网络成绩是: 90
```

```
我的形式语言与自动机成绩是: 85
```



字典的应用：产生多分支

```
def f1():  
    print('f1 is executed')  
    return  
  
def f2():  
    print('f2 is executed')  
    return  
  
def f3():  
    print('f3 is executed')  
    return  
  
if __name__ == '__main__':  
    d = {1:f1, 2:f2, 3:f3}    #f1,f2,f3为3个函数的名字  
    i = 1  
    d[i]()                  #根据i的值，选择不同分支，这里执行f1()
```



字典应用的典型案例：词频统计

统计Hamlet中各单词出现的频率，输出出现次数最多的10个单词及其出现的个数

```
hamlet - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
hamlet.txt

The Tragedy of Hamlet, Prince of Denmark
Shakespeare homepage | Hamlet | Entire play
ACT I

SCENE I. Elsinore. A platform before the castle.

FRANCISCO at his post. Enter to him BERNARDO
BERNARDO
Who's there?
FRANCISCO
Nay, answer me: stand, and unfold yourself.
BERNARDO
Long live the king!
```




字典应用的综合案例

```
def getText():  
    txt = open("hamlet.txt", 'r').read()  
    txt = txt.lower()    #将所有文本中的英文全部换为小写字母  
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':  
        txt = txt.replace(ch, ' ')    #将文本中的特殊字符替换为空格  
    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)  
print(' word      count')  
for i in range(10):  
    word, count = items[i]  
    print(f'{word:<12}{count:>6}')
```

函数getText():
读入txt文件, 进行预处理

1. 读入txt文件, 进行预处理
2. 对文本进行分词
3. 对分词后的列表统计词频, 写入字典
4. 将词频字典转换为列表, 按值排序
5. 输出



字典应用的综合案例：使用列表

```
def getText():
    txt = open("hamlet.txt", 'r').read()
    txt = txt.lower()    #将所有文本中的英文全部换为小写字母
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
        txt = txt.replace(ch, ' ')    #将文本中的特殊字符替换为空格
    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)

print(' word      count')
for i in range(10):
    word, count = items[i]
    print(f' {word:<12} {count:>6}')
```

counts[[word1,num1],...]

```
for word in words:
    for count in counts:
        if count[0] == word:
            count[1] += 1
            break
    else:
        counts.append([word,0])

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



字典应用的综合案例：时间对比

```
import time

def getText():
    txt = open("hamlet.txt", 'r').read()
    txt = txt.lower()    #将所有文本中的英文全部换为小写字母
    for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
        txt = txt.replace(ch, ' ')    #将文本中的特殊字符替换为空格
    return txt
```

```
starttime=time.time()
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)
endtime=time.time()
print('time:',endtime-starttime)
print(' word      count')
for i in range(10):
    word, count = items[i]
    print(f'{word:<12}{count:>6}')
```

time: 0.02437448501586914

time: 1.8596348762512207



字典应用的典型案例：中文词频统计

统计《明朝那些事儿》中词出现的频率，输出出现次数最多的10个词及其出现的个数

明朝那些事儿 - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

《明朝那些事儿》

作者：当年明月

前言

好了，今天晚上开始工作吧！

说起来，我也写了不少东西了，主要是心理和历史方面的，偶尔也写点经济，本来只是娱乐下自己，没有想到发表后居然

也是这件事，让我反思了自己的行为和态度，明白了自己其实还差得远。所以我希望能重新开始，下面的这篇文章我构思

我写文章有个习惯，由于早年读了太多学究书，所以很痛恨那些故作高深的文章，其实历史本身很精彩，所有的历史都可

其实我也不知道自己写的算什么，不是小说，不是史书，就姑且叫《明札记》吧，从我们的第一位主人公写起，要写三百

2006-3-10

明朝那些事儿.txt



字典应用的典型案例：中文词频统计

```
starttime=time.time()
txt = getText()
words = jieba.lcut(txt)
counts = {}

stopwords1 = [line.rstrip() for line in open('中文停用词表.txt', 'r', encoding='utf-8')]
stopwords2 = [line.rstrip() for line in open('百度停用词表.txt', 'r', encoding='utf-8')]
stopwords3 = [line.rstrip() for line in open('哈工大停用词表.txt', 'r', encoding='utf-8')]
stopwords4 = [line.rstrip() for line in open('四川大学停用词表.txt', 'r', encoding='utf-8')]
stopwords = stopwords1 + stopwords2 + stopwords3 + stopwords4

for word in words:
    if word in stopwords or len(word) == 1:
        continue #不希望统计到单个词, 比如说“的”, “好”等
    counts[word] = counts.get(word, 0)+1

items = list(counts.items())
items.sort(key=lambda x:x[1], reverse=True)
endtime=time.time()
print('time:',endtime-starttime)
print(' word          count')
for i in range(10):
    word, count = items[i]
    print(f'{word:<12}{count:>6}')
```

time: 21.969829320907593

word	count
皇帝	2093
朱棣	1415
事情	1340
朱元璋	1290
这位	1271
实在	1152
终于	1017
这是	746
万历	725
明朝	657



字典应用的典型案例：中文词频统计

```
stopwords1 = [line.rstrip() for line in open('中文停用词表.txt', 'r', encoding='utf-8')]
stopwords2 = [line.rstrip() for line in open('百度停用词表.txt', 'r', encoding='utf-8')]
stopwords3 = [line.rstrip() for line in open('哈工大停用词表.txt', 'r', encoding='utf-8')]
stopwords4 = [line.rstrip() for line in open('四川大学停用词表.txt', 'r', encoding='utf-8')]
stopwords = stopwords1 + stopwords2 + stopwords3 + stopwords4
```

```
stopwords1 = [line.rstrip() for line in open('中文停用词表.txt', 'r', encoding='utf-8')]
stopwords2 = [line.rstrip() for line in open('百度停用词表.txt', 'r', encoding='utf-8')]
stopwords3 = [line.rstrip() for line in open('哈工大停用词表.txt', 'r', encoding='utf-8')]
stopwords4 = [line.rstrip() for line in open('四川大学停用词表.txt', 'r', encoding='utf-8')]
stopwords5 = ['事情', '这位', '实在', '终于', '这是', '皇帝', '大臣', '明朝', '两个', '朝廷', '地方']
stopwords = stopwords1 + stopwords2 + stopwords3 + stopwords4 + stopwords5
```

word	count
朱棣	1414
朱元璋	1289
万历	724
张居正	633
徐阶	616
明军	597
嘉靖	593
严嵩	564
崇祯	562
王守仁	535



字典应用的综合案例：时间对比

使用字典

```
for word in words:
    if word in stopwords or len(word) == 1:
        continue
    counts[word] = counts.get(word, 0) + 1
items = list(counts.items())
items.sort(key=lambda x: x[1], reverse=True)
```

time: 21.803346395492554

使用列表

```
for word in words:
    if word in stopwords or len(word) == 1:
        continue
    for count in counts:
        if count[0] == word:
            count[1] += 1
            break
    else:
        counts.append([word, 0])
counts.sort(key=lambda x: x[1], reverse=True)
```

time: 255.76704692840576



组合数据类型：总结

- 列表、字符串、元组属于有序序列，支持双向索引；集合、字典属于无序序列，集合不支持使用下标的方式访问其中的元素，可以使用字典的键（key）作为下标访问字典中的值（value）
- 列表、字典、集合属于可变组合，可以进行增删改操作；元组、字符串属于不可变组合，不能进行增删改操作（元组存在例外）
- 列表（元组、集合、字典）推导式可以用简洁的方式生成满足特定要求的列表（元组、集合、字典）
- 对包括字符串在内的所有组合数据类型都可以执行`any()`、`all()`操作，为空的组合的`all()`值为`True`，`any()`值为`False`



组合数据类型：总结

- 对包括字符串在内的所有组合数据类型都可以执行in、not in操作，但字典类型仅针对键（key）作出判断
- 列表、元组中的元素可以是任意数据类型，且可以各不相同
- 集合中元素的数据类型必须是不可变数据类型（可哈希），且元素唯一
- 字典中元素的键必须是不可变数据类型，且键唯一；值可以是任意类型



组合数据类型：总结

- 列表元素增加应优先使用`append()`、`extend()`操作，不建议使用“+”、“*”或`insert()`
- 切片操作不但可以返回列表、元组、字符串中的部分元素，还可以对列表中的元素进行增删改
- 对字典元素的访问应优先使用`get()`