组合数据类型（2）

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

- 不能对元组元素进行增删改

```
>>> tup2[1]=5
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    tup2[1]=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 = ()
>>> tup2
()
```

```
>>> tup3 = 1, 'a', True, -2
>>> tup3
(1, 'a', True, -2)
```

```
>>> tup5 = (1,)
>>> tup5
(1,)
```

```
>>> tup5 = 1,
>>> tup5
(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

```
>>> tup3 = tuple([1,3,5,7])
>>> tup3
(1, 3, 5, 7)

>>> tup4 = tuple(range(2,8,2))
>>> tup4
(2, 4, 6)

>>> tup5 = tuple('Hello')
>>> tup5
('H', 'e', 'l', 'l', 'o')

>>> tup6=tuple(x**2 for x in range(5))
>>> tup6
(0, 1, 4, 9, 16)
```

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

  tuple[i]

```
>>> tup3 = (1,3,5,7,9)
>>> x, y = tup3[2], tup3[-1]
>>> print(x,y)
5 9
```

- 不能通过索引进行元组元素的修改

```
>>> tup3 = (1,3,5,7,9)
>>> tup3[2] = -1
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    tup3[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

```
>>> del tup3[4]
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    del tup3[4]
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

tuple[start:stop:step]

```
>>> tup3
(1, 3, 5, 7, 9)
>>> x, y = tup3[0:3:2], tup3[-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

```
>>> tup3 = (1,3,5,7,9)
>>> tup3[0:3:2] = (-1, -6)
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    tup3[0:3:2] = (-1, -6)
TypeError: 'tuple' object does not support item assignment


>>> del tup3[0:4:2]
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    del tup3[0:4:2]
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

tuple.index(x[,i[,j]])

```
>>> tup4 = (1, 2, 1, 3, 5, 1, 7, 9, 11)
>>> tup4.index(1)
0
```

```
>>> tup4.index(1,2)
2
```

```
>>> tup4.index(1,4,8)
5
```

```
>>> tup4.index(1,6)
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    tup4.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 tuple

```
>>> tup3
(1, 3, 5, 7, 9)
>>> del tup3

>>> tup3
Traceback (most recent call last):
  File "<pyshell#63>", line 1, in <module>
    tup3
NameError: name 'tup3' is not defined
```

元组的成员判断操作

- 使用in判断元素是否在元组中

  x in tuple

- 使用 not in判断元素是否不在元组中

  x not in tuple

```
>>> tup3 = (1,'a',5)
>>> 'a' in tup3
True
>>> 8 in tup3
False
>>> 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#11>", line 1, in <module>
    (1,2)<(1,'a')
TypeError: unorderable types: int() < str()

>>> t1 = (3,"abc",4)
>>> t2 = (3,"abc",2)
>>> t3 = (3,"abcc",2)
>>> t4 = (3,"abc",4,"c")
>>> print(t1 == t2, t1>t2, t1<t2)
False True False

>>> print(t1 == t3, t1>t3, t1<t3)
False False True

>>> print(t1 == t4, t1>t4, t1<t4)
False False True
```

元组的统计操作

函数或方法	描述
len(tuple)	返回元组中元素的个数
max(tuple)	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(tuple)	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
tuple.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(tuple)

- 通过any()函数判断元组中是否有值为真

  any(tuple)

```
>>> tup3 = (1,'a',-1,True)
>>> all(tup3)
True
>>> any(tup3)
True
```

```
>>> tup4 = ("",1)
>>> all(tup4)
False
>>> any(tup4)
True
```

```
>>> tup5=()
>>> all(tup5)
True
>>> any(tup5)
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(tuple, key = None, reverse = False)

```
>>> tup1 = (2,8,-2,0)
>>> sorted(tup1)
[-2, 0, 2, 8]

>>> tup1 = (2,8,-2,0)
>>> tup2 = tuple(sorted(tup1))
>>> tup2
(-2, 0, 2, 8)

>>> sorted(tup1,key=k1,reverse=True)
[8, -2, 0, 2]
```

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

```
i= 2 k1(i)= 0
i= 8 k1(i)= 36
i= -2 k1(i)= 16
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

  tuple = tuple + (x, y...)

- 通过*在元组中增加重复元素

  tuple = tuple * n

- 通过切片或切片和+的组合在元组中删除元素

  tuple = tuple[start1:stop1:step1] +tuple[start2:stop2:step2]

```
>>> tup4 = (1,3,5,7)
>>> tup4 = tup4[1:4]
>>> tup4
(3, 5, 7)
```

```
>>> tup5 = (1,3,5,7,9,11)
>>> tup5 = tup5[:3]+tup5[4:]
>>> tup5
(1, 3, 5, 9, 11)
```

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)
34783400

>>> a[1][0]=-5
>>> a
(1, [-5, 3])
>>> id(a)
34783400
```

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
list.append(x)	将x添加到列表末尾，列表长度加1
list.insert(i,x)	在序号为i的位置插入对象x，i以后的元素后移
list.extend(t)	把可迭代对象t附加到s的尾部
list.copy()	复制出一个新列表，返回给调用者，复制过程为浅拷贝
del list[i]	删除序号为i的元素，i以后的元素前移
list.pop[(i])	删除列表末尾或序号为i的位置的元素，返回该元素的值
list.remove(x)	删除列表中第一次出现的x值
list.clear()	清空列表元素
del list	删除列表
list.sort()	对列表排序，修改列表
sorted(list)	对列表排序，生成新列表
list.reverse()	将列表反序，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 = {(x-2)**2 for x in range(5)}
>>> s7
{0, 1, 4}
```

- 直接赋值

```
>>> s = {1,2,3,'ok',1,(0,6)}
>>> s
{1, 2, 3, (0, 6), 'ok'}
```

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

- 通过set()函数将其他类型的数据转换为集合

```
>>> s5 = set((1,2,'ok'))
>>> s5
{1, 2, 'ok'}
```

```
>>> s3 = set([1,3,3,5,7,1])
>>> s3
{1, 3, 5, 7}
```

```
>>> s4 = set(range(2,8,2))
>>> s4
{2, 4, 6}
```

```
>>> s6 = set("Hello")
>>> s6
{'o', 'H', 'l', 'e'}
```

集合的基本操作：集合元素的增加

- add()方法

set.add(x)

```
>>> s4
{2, 4, 6}
>>> s4.add(-1)
>>> s4
{2, -1, 4, 6}
>>> s4.add(2)
>>> s4
{2, -1, 4, 6}
```

```
>>> s4.add('abc')
>>> s4
{2, 'abc', 4, 6, -1}

>>> s4.add(('A',8))
>>> s4
{2, 'abc', 4, 6, ('A', 8), -1}
```

```
>>> s4.add([0,7])
Traceback (most recent call last):
  File "<pyshell#163>", line 1, in <module>
    s4.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
{2, -1, 4, 6}
>>> s4.update({0,3,5,4})
>>> s4
{0, 2, 3, 4, 5, 6, -1}
>>> s4.update([1])
>>> s4
{0, 1, 2, 3, 4, 5, 6, -1}
```

```
>>> s4.update("abc")
>>> s4
{0, 1, 2, 3, 4, 5, 6, 'b', 'c', -1, 'a'}
```

```
>>> s4.update((1,[2,3]))
Traceback (most recent call last):
  File "<pyshell#54>", 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

set.remove(x)

```
>>> s3
{0, -3, 5, True, 'a'}
>>> s3.remove(5)

>>> s3
{0, -3, True, 'a'}
```

```
>>> s3.remove(5)
Traceback (most recent call last):
  File "<pyshell#206>", line 1, in <module>
    s3.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

set.discard(x)

```
>>> s4 = {'a','b','c','d'}
>>> s4.discard('b')
>>> s4
{'c', 'd', 'a'}
```

```
>>> s4.discard('A')
>>> s4
{'c', 'd', '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

set.pop()

```
>>> s4 = {'a','b','c','d'}
>>> s4.pop()
'c'
>>> s4
{'b', 'd', 'a'}
```

```
>>> s4.pop()
'b'
>>> s4.pop()
'd'
>>> s4.pop()
'a'
>>> s4.pop()
Traceback (most recent call last):
  File "<pyshell#224>", line 1, in <module>
    s4.pop()
KeyError: 'pop from an empty set'
>>> s4
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

  set.clear()

```
>>> s4 = {'a','b','c','d'}
>>> s4.clear()
>>> s4
set()
```

- 使用del删除集合

  del set

```
>>> s3
{-3, True, 'a'}
>>> del s3

>>> s3
Traceback (most recent call last):
  File "<pyshell#229>", line 1, in <module>
    s3
NameError: name 's3' is not defined
```

集合的成员判断操作

- 使用in判断元素是否在集合中

  x in set

- 使用 not in判断元素是否不在集合中

  x not in set

```
>>> s4 = {'a','b','c','d'}
>>> 'a' in s4
True
>>> 'dd' in s4
False
```

```
>>> 'cc' not in s4
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(set)	返回集合中元素的个数
max(set)	返回集合元素中的最大值，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(set)	返回集合元素中的最小值，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(set)	判断集合中元素是否全为True
any(set)	判断集合中是否有元素为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(set, key = None, reverse = False)

```
>>> s1 = {2,8,-2,0}
>>> sorted(s1)
[-2, 0, 2, 8]

>>> s1 = {2,8,-2,0}
>>> s2 = set(sorted(s1))
>>> s2
{0, 8, 2, -2}

>>> sorted(s1,key=k1,reverse=True)
[8, -2, 0, 2]
```

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

i= 2 k1(i)= 0
i= 8 k1(i)= 36
i= -2 k1(i)= 16
i= 0 k1(i)= 4
```

集合的复制操作

- 赋值

```
set2 = set1
```

- 浅复制

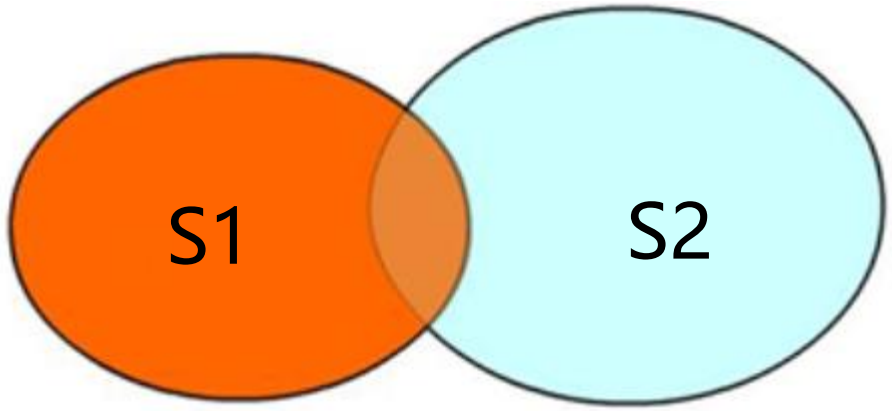
```
set2 = set1.copy()
set2 = copy.copy(set1)
```

- 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^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
s1 == s2	若s1和s2长度相等，元素完全相同，则返回True;否则返回False
s1 != s2	若s1和s2元素不完全相同，则返回True;否则返回False
s1 > s2	若s1是s2的纯超集，则返回True;否则返回False
s1 < s2	若s1是s2的纯子集，则返回True;否则返回False
s1 >= s2	若s1是s2的超集，则返回True;否则返回False
s1 <= s2	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
s1.isdisjoint(s2)	如果s1和s2之间没有共同元素，返回True，否则False
s1.issubset(s2)	如果s1是s2的子集，返回True，否则False
s1.isuperset(s2)	如果s1是s2的超集，返回True，否则False
s1.union(s2,...)	返回s1和s2、...的并集
s1.intersection(s2,...)	返回s1和s2、...的交集
s1.difference(s2,...)	返回s1-s2的差集
s1.symmetric_difference(s2)	返回s1和s2的对称差

集合运算示例

```
>>> s1 = {1,2,3}
>>> s2 = {1,4,5}
>>> s3 = {5,8}

>>> print(s1-s2)
{2, 3}

>>> print(s1|s3)
{1, 2, 3, 5, 8}

>>> print(s2^s3)
{8, 1, 4}
>>> print(s2>s1)
False
```

```
>>> print(s1.union(s3,s2))
{1, 2, 3, 4, 5, 8}
>>> print(s2.difference(s3))
{1, 4}
>>> print(s1.intersection(s2,s3))
set()
>>> print(s2.isdisjoint(s3))
False
>>> print(s3.issubset(s2))
False
```

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()

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

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

```
>>> fs3 = frozenset("Hello world")
>>> fs3
frozenset({'o', ' ', 'l', 'r', 'w', 'd', 'H', 'e'})
```

不能用于不可变集合的函数或方法

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

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

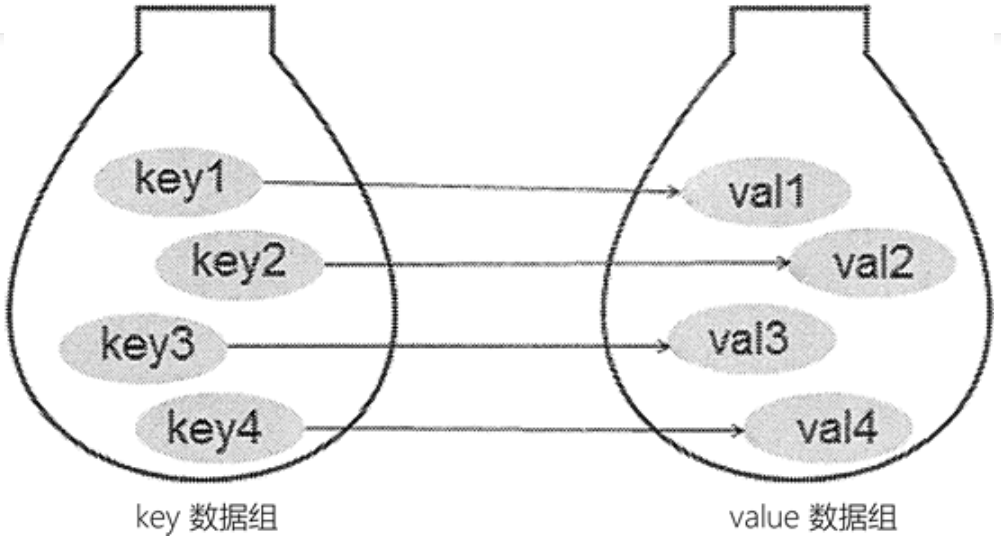
中文姓名	利昂内尔·安德烈斯·梅西·库西蒂尼
外文姓名	Lionel Andrés Messi Cuccitini
国籍	阿根廷
出生地	阿根廷圣菲省罗萨里奥
出生日期	1987年6月24日
身高	1.7m
体重	72kg

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：

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

- 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

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

- 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

字典的基本操作：字典创建

- 直接赋值

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

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

- 通过dict()函数生成字典

```
>>> items =[('name','Mary'),('age',21)]
>>> dict3 = dict(items)
>>> dict3
{'name': 'Mary', 'age': 21}
```

```
>>> dict4 = dict(name='Mary',age=21,height=1.67)
>>> dict4
{'name': 'Mary', 'height': 1.67, 'age': 21}
```

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 = 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
```

Python 3.10.0

Release Date: Oct. 4, 2021

字典的基本操作：字典创建

- 通过fromkeys()函数创建初始字典

```
>>> dict6 = dict.fromkeys(countries)

>>> dict6
{'China': None, 'United States': None, 'Russia': None, 'United Kingdom':
None, 'France': None}

>>> dict6 = dict.fromkeys(countries, 'Beijing')
>>> dict6
{'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

dict[k] = value

- 在字典中新增一个元素

```
>>> score
{'计算机网络': 86, '计算机组成原理': 90}
>>> score['形式语言与自动机'] = 88
>>> score
{'计算机网络': 86, '形式语言与自动机': 88, '计算机组成原理': 90}
```

- 在字典中修改一个元素

```
>>> score['形式语言与自动机'] = 96
>>> score
{'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
```

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 dict[k]

```
>>> score
{'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> del score['计算机网络']
>>> score
{'形式语言与自动机': 96, '计算机组成原理': 90}
```

- 在字典中删除一个元素，并返回被删除的值　　val =dict.pop(k)

```
>>> sc1 = score.pop('形式语言与自动机')
>>> score
{'计算机组成原理': 90}
>>> sc1
96
```

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

```
>>> sc1 = score.pop('形式语言与自动机')
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    sc1 = 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': 90}
>>> del score['计算机网络']
Traceback (most recent call last):
  File "<pyshell#13>", 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

  dict.pop(k [,default])

```
>>> score
{'计算机组成原理': 90}
>>> sc1 = score.pop('形式语言与自动机', -1)
>>> score
{'计算机组成原理': 90}
>>> sc1
-1
```

- 删除前通过操作符in进行判断

```
>>> '计算机网络' in score
False
```

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

dict.clear()

```
>>> score = {'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> score
{'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> score.clear()
>>> score
{}
```

- 使用del删除字典

del dict

```
>>> score = {'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> score
{'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> del score
>>> score
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    score
NameError: name 'score' is not defined
```

字典的基本操作：增删改

函数或方法	描述
dict[k] = v	在字典中增加元素 k:v
dict[k] = nv	将字典中键k对应的值修改为nv，即将该元素修改为 k:nv
del dict[k]	删除字典中键k对应的元素
dict.pop (k[, default])	删除字典中键k对应的元素，并返回被删除的值
dict.clear()	删除字典中的所有元素
del dict	删除字典

访问字典中的值：避免错误

- 直接访问字典中的值　　dict[k]

```
>>> score
{'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> score['数据结构程序设计']
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    score['数据结构程序设计']
KeyError: '数据结构程序设计'
```

- 使用get()函数

dict.get(k[,default])

```
>>> score
{'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> sc2 = score.get('计算机网络')
>>> sc2
86
>>> sc3 = score.get('数据结构程序设计',-1)
>>> sc3
-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
dict.items()	获得字典中所有的键值对
dict.keys()	获得字典中所有的键
dict.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
{'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
>>> score.items()
dict_items([('计算机网络', 86), ('形式语言与自动机', 96), ('计算机组成原理', 90)])
>>> score.keys()
dict_keys(['计算机网络', '形式语言与自动机', '计算机组成原理'])
>>> score.values()
dict_values([86, 96, 90])
```

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 = {'计算机网络': 86, '形式语言与自动机': 96, '计算机组成原理': 90}
for key, value in score.items():
    print("My score of {} is {}".format(key,value))
```

```
My score of 计算机网络 is 86
My score of 计算机组成原理 is 90
My score of 形式语言与自动机 is 96
```

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()
```

通过match产生多分支

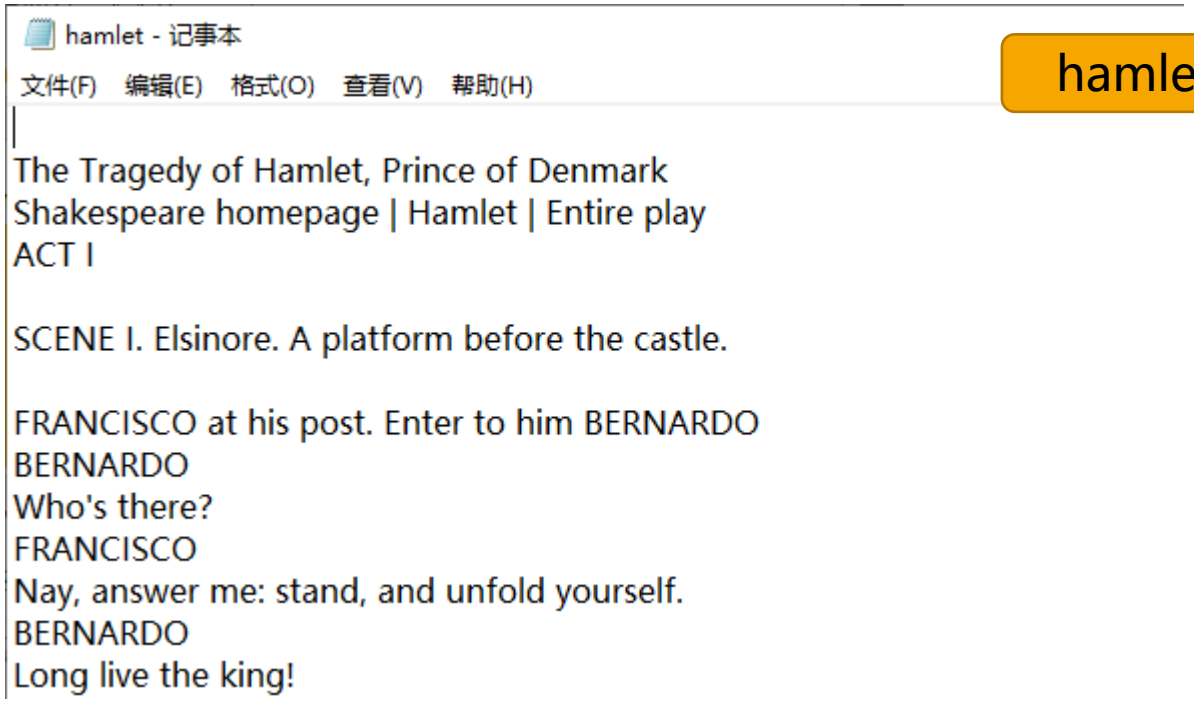
```
def f1():
    print('f1 is executed')
    return

def f2():
    print('f2 is executed')
    return

def f3():
    print('f3 is executed')
    return

if __name__ =='__main__':
    i=1
    match i:
        case 1: f1()
        case 2: f2()
        case 3: f3()
```

Python 3.10.0

Release Date: Oct. 4, 2021

字典应用的典型案例：词频统计

统计Harmlet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.txt

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

```
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}')
```

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

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
    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],...]

```
counts = []
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}')
```

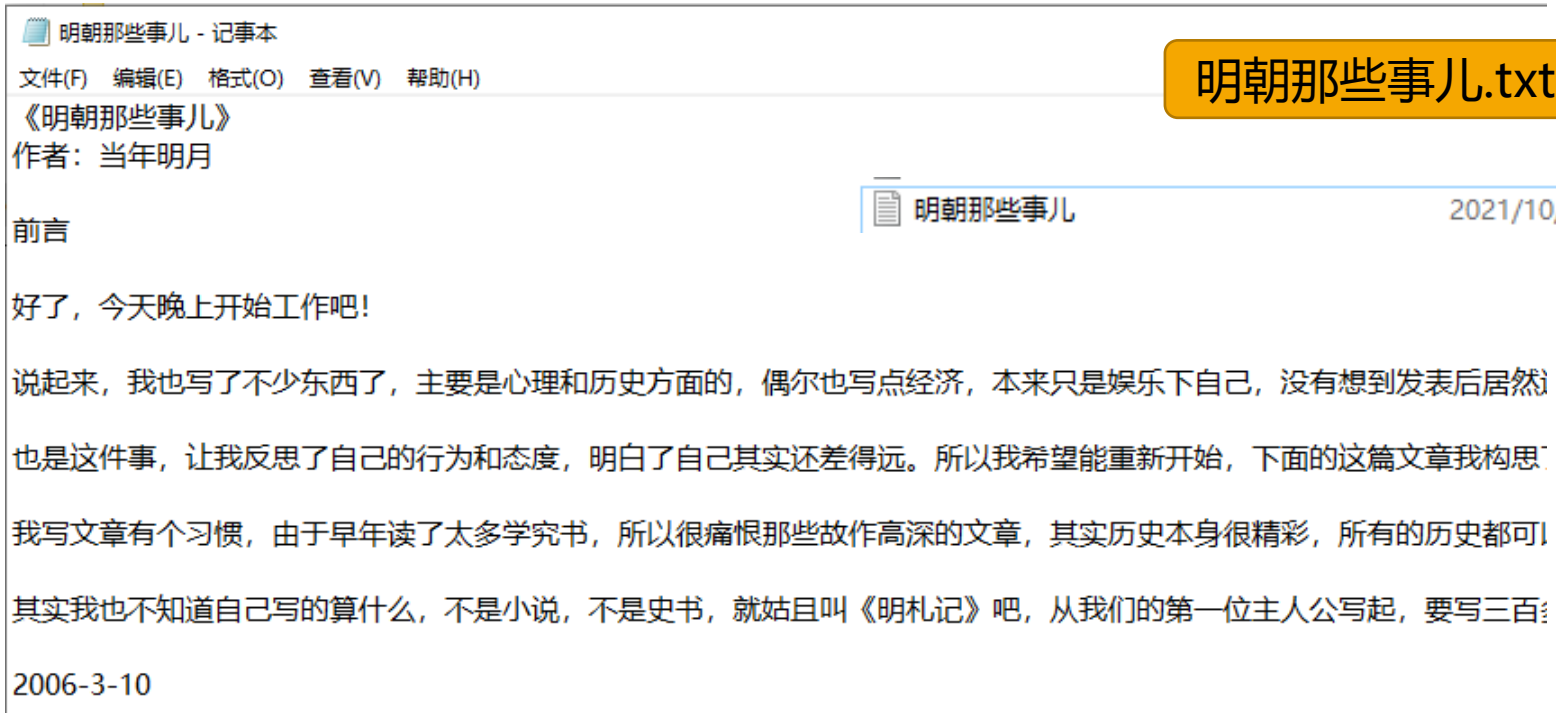
```
time: 0.02437448501586914
```

```
time: 1.859634876251220 7
```

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.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: 38.56686568260193
 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: 38.56686568260193

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: 403.09658622741

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需要的列表（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()