**namedtuple**

映射名称到序列元素，替代字典，节省空间

Stock **=** namedtuple('Stock', ['name', 'shares', 'price'])

**def** **compute\_cost**(records):

total **=** 0.0

**for** rec **in** records:

s **=** Stock(**\***rec)

total **+=** s**.**shares **\*** s**.**price

**return** total

#命名元组不可更改，修改需要\_replace()

>>> s = Stock('ACME', 100, 123.45)

>>> s **=** s**.**\_replace(shares**=**75)

>>> s

Stock(name='ACME', shares=75, price=123.45)

#\_replace可以用来填充数据

Stock **=** namedtuple('Stock', ['name', 'shares', 'price', 'date', 'time'])

stock\_prototype **=** Stock('', 0, 0.0, None, None)

**def** **dict\_to\_stock**(s):

**return** stock\_prototype**.**\_replace(**\*\***s)

>>> a **=** {'name': 'ACME', 'shares': 100, 'price': 123.45}

>>> dict\_to\_stock(a)

Stock(name='ACME', shares=100, price=123.45, date=None, time=None)

>>> b **=** {'name': 'ACME', 'shares': 100, 'price': 123.45, 'date': '12/17/2012'}

>>> dict\_to\_stock(b)

Stock(name='ACME', shares=100, price=123.45, date='12/17/2012', time=None)

**deque**

使用list存储数据时，按索引访问元素很快，但是插入和删除元素就很慢了，因为list是线性存储，数据量大的时候，插入和删除效率很低。

deque是为了**高效实现插入和删除操作的双向列表**，适合用于队列和栈：

class deque(builtins.object)

| deque([iterable[, **maxlen**]]) --> deque object  **# maxlen为list最大长度**

| append(...)

| Add an element to the right side of the deque.

|  **appendleft(...)**

| Add an element to the left side of the deque.

| clear(...)

| Remove all elements from the deque.

| copy(...)

| Return a shallow copy of a deque.

| count(...)

| D.count(value) -> integer -- return number of occurrences of value

| extend(...)

| Extend the right side of the deque with elements from the iterable

|  **extendleft(...)**

| Extend the left side of the deque with elements from the iterable

| index(...)

| D.index(value, [start, [stop]]) -> integer -- return first index of value.

| Raises ValueError if the value is not present.

| insert(...)

| D.insert(index, object) -- insert object before index

| pop(...)

| Remove and return the rightmost element.

| popleft(...)

| Remove and return the leftmost element.

| remove(...)

| D.remove(value) -- remove first occurrence of value.

| reverse(...)

| D.reverse() -- reverse \*IN PLACE\*

**| rotate(...)**

| Rotate the deque n steps to the right (default n=1). If n is negative, rotates left.

保留有限历史记录（利用collections.deque）

>>> from collections import deque

>>> def search(lines,pattern,history=5):

previous\_lines=deque(maxlen=history)

for li in lines:

if pattern in li:

yield li, previous\_lines

previous\_lines.append(li)

### defaultdict(list|set|int)（类似于D.get()）

### #自动初始化每个key 刚开始对应的值，避免KeyError；便于一键多值

**from collections import** defaultdict

d = defaultdict(list)

d['a'].append(1)

d['a'].append(2)

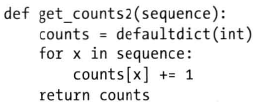
d['b'].append(4)

d = defaultdict(set)

d['a'].add(1)

d['a'].add(2)

d=defaultdict(int) #所有值均会被初始化为0



### OrderedDict

#保持Key插入顺序

**rom collections import** OrderedDict

**def** ordered\_dict():

d = OrderedDict()

d['foo'] = 1

d['bar'] = 2

d['spam'] = 3

d['grok'] = 4

# Outputs "foo 1", "bar 2", "spam 3", "grok 4"

**>>> import json**

**>>>** json.dumps(d)

'{"foo": 1, "bar": 2, "spam": 3, "grok": 4}'

# OrderedDict 内部维护着一个根据键插入顺序排序的双向链表，因此一个OrderedDict 的大小是一个普通字典的两倍

### Counter

Counter是一个简单的计数器

| elements(self)

| Iterator over elements repeating each as many times as its count.

|

| >>> c = Counter('ABCABC')

| >>> sorted(c.elements())

| ['A', 'A', 'B', 'B', 'C', 'C']

|

| **most\_common**(self, **n**=None)

| **List the n most common elements and their counts from the most**

| common to the least. If n is None, then list all element counts.

|

| >>> Counter('abcdeabcdabcaba').most\_common(3)

| [('a', 5), ('b', 4), ('c', 3)]

|

| **subtract**(\*args, \*\*kwds)

| Like dict.update() but subtracts counts instead of replacing them.

| **Counts can be reduced below zero**. Both the inputs and outputs are

| allowed to contain zero and negative counts.

|

| >>> c = Counter('which')

| >>> c.subtract('witch') # subtract elements from another iterable

| >>> c.subtract(Counter('watch')) # subtract elements from another counter

| >>> c['h'] # 2 in which, minus 1 in witch, minus 1 in watch

| 0

| >>> c['w'] # 1 in which, minus 1 in witch, minus 1 in watch

| -1

|

| **update**(\*args, \*\*kwds)

| Like dict.update() but **add** counts instead of replacing them.

|

| >>> c = Counter('which')

| >>> c.update('witch') # add elements from another iterable

| >>> d = Counter('watch')

| >>> c.update(d) # add elements from another counter

| >>> c['h'] # four 'h' in which, witch, and watch

| 4

#可以很容易的跟数学运算操作相结合

**>>>** a = Counter(words)

**>>>** b = Counter(morewords)

**>>>** a

Counter({'eyes': 8, 'the': 5, 'look': 4, 'into': 3, 'my': 3, 'around': 2, "you're": 1, "don't": 1, 'under': 1, 'not': 1})

**>>>** b

Counter({'eyes': 1, 'looking': 1, 'are': 1, 'in': 1, 'not': 1, 'you': 1, 'my': 1, 'why': 1})

**>>>** *# Combine counts*

**>>>** c = a + b

**>>>** c

Counter({'eyes': 9, 'the': 5, 'look': 4, 'my': 4, 'into': 3, 'not': 2, 'around': 2, "you're": 1, "don't": 1, 'in': 1, 'why': 1, 'looking': 1, 'are': 1, 'under': 1, 'you': 1})

**>>>** *# Subtract counts*

**>>>** d = a - b

**>>>** d

Counter({'eyes': 7, 'the': 5, 'look': 4, 'into': 3, 'my': 2, 'around': 2, "you're": 1, "don't": 1, 'under': 1})

### chainMap

将多个字典或者映射从逻辑上合并为一个单一的映射

a **=** {'x': 1, 'z': 3 }

b **=** {'y': 2, 'z': 4 }

c **=** ChainMap(a,b)

**print**(c['x']) *# Outputs 1 (from a)*

**print**(c['y']) *# Outputs 2 (from b)*

**print**(c['z']) *# Outputs 3 (from a)*

#如果出现重复键，那么第一次出现的映射值会被返回