collections

一、namedtuple

我们知道 tuple 可以表示不变集合，例如，一个点的二维坐标就可以表示成：

```
>>> p = (1, 2)
```

但是，看到(1, 2)，很难看出这个 tuple 是用来表示一个坐标的，而定义一个 class 又小题大做了，这时，namedtuple 就派上了用场：

```
>>> from collections import namedtuple
>>> Point = namedtuple("Point",["x","y"])
>>> p = Point(1,2)
>>> p.x
1
>>> p.y
2
```

namedtuple 是一个函数，它用来创建一个自定义的 tuple 对象，并且规定了 tuple 元素的个数，并可以用属性而不是索引来引用 tuple 的某个元素。

这样一来，我们用 namedtuple 可以很方便地定义一种数据类型，它具备 tuple 的不变性，又可以根据属性来引用，使用十分方便。

可以验证创建的 Point 对象是 tuple 的一种子类：

```
>>> isinstance(p,Point)
True
>>> isinstance(p,tuple)
True
```

namedtuple 函数定义

namedtuple(typename, field_names, *, rename=False, defaults=None, module=None)

参数：

typename：	类型名
field_names：	属性列表
rename ：	如果为真，则无效域名会自动转换为位置名，比如 ['abc', 'def', 'ghi', 'abc'] 转换成['abc', '_1', 'ghi', '_3']，消除关键词 def 和重复域名 abc。
defaults：	可以为 None 或者是一个默认值的 iterable。如果一个默认值域必须跟其他没有默认值的域在一起出现，defaults 就应用到最右边的参数。比如如果域名['x', 'y', 'z']和默认值(1, 2)，那么 x 就必须指定一个参数值，y 默认值 1，z 默认值 2。
module：	如果 module 值有定义，命名元组的__module__属性值就被设置。

再举一个例子，创建一个圆：

```
>>> from collections import namedtuple
>>> Circle = namedtuple("Circle",['x','y','r'])
>>> c = Circle(0,0,1)
>>> c
Circle(x=0, y=0, r=1)
```

二、deque

deque 是一个双向链表，针对 list 连续的数据结构插入和删除进行优化。deque 是一个类，下面是他的构造方法：

deque([iterable[, maxlen]]) --> deque object

接受一个可迭代对象（内置的就是 str、tuple、list 咯），第二个参数指定最大队列长度（maxlen）

双向队列(deque)对象支持以下方法：

1. append(x)

添加 x 到右端

2. appendleft(x)

添加 x 到左端

3. clear()

移除所有元素，使其长度为 0

4. copy()

创建一份浅拷贝

5. count(x)

计算 deque 中个数等于 x 的元素

6. extend(iterable)

扩展 deque 的右侧，通过添加 iterable 参数中的元素

7. extendleft(iterable)

扩展 deque 的左侧，通过添加 iterable 参数中的元素。注意，左添加时，在结果中 iterable 参数中的顺序将被反过来添加

8. index(self, value, start=None, stop=None)

在给定范围内查找 value，返回第一个匹配的索引，如果没找到的话报错

9. insert(index, p_object)

在 index 处插入 object，如果插入会导致一个限长 deque 超出长度 maxlen 的话，就报错

10. pop()

移去并且返回一个元素，deque 最右侧的那一个。如果没有元素的话，报错

11. popleft()

移去并且返回一个元素，deque 最左侧的那一个。如果没有元素的话，报错

12. remove(value)

删除第一次出现的 value 元素，value 不存在就报错

13. reverse()

逆置双端队列

14. rotate(n = 1)

向右循环移动 n 步。如果 n 是负数，就向左循环。如果 deque 不是空的，向右循环移动一步就等价于 d.appendleft(d.pop())，向左循环一步就等价于 d.append(d.popleft())

15. 只读属性：maxlen

Deque 的最大尺寸，如果没有限定的话就是 None

三、defaultdict

使用 dict 时，如果引用的 Key 不存在，就会抛出 KeyError。如果希望 key 不存在时，返回一个默认值，就可以用 defaultdict，构造器语法为：

defaultdict([default_factory[, ...]])

其中，第一个参数是一个函数，它是在你想得到一个不存在的 key 的 value 时，自动创建一个 value 返回。

其他方法和普通 dici 基本一致。

四、OrderedDict

使用 dict 时，Key 是无序的。在对 dict 做迭代时，我们无法确定 Key 的顺序。如果要保持 Key 的顺序，可以用 OrderedDict，注意，OrderedDict 的 Key 会按照插入的顺序排列，不是 Key 本身排序：

```
>>> from collections import OrderedDict
>>> od = OrderedDict()
>>> od['z'] = 1
>>> od['y'] = 2
>>> od['x'] = 3
>>> list(od.keys())
['z', 'y', 'x']
```

OrderedDict 可以实现一个 FIFO（先进先出）的 dict，也可以实现 LIFO 的栈。它的方法和普通 dict 基本一致，主要注意下面 2 个方法：

1. popitem(last=True)

有序字典的 popitem() 方法移除并返回一个 (key, value) 键值对。 如果 last 值为真，则按 LIFO 后进先出的顺序返回键值对，否则就按 FIFO 先进先出的顺序返回键值对。

2. move_to_end(key,last=True)

将现有 key 移动到有序字典的任一端。 如果 last 为真值（默认）则将元素移至末尾；如果 last 为假值则将元素移至开头。如果 key 不存在则会触发 KeyError。

五、Counter

Counter 是 dict 的子类，所以操作同 dict，在此基础上，又添加了 most_common(),elements()。

1. Counter([iterable-or-mapping])

```
>>> c = Counter()                           # new, empty counter
>>> c = Counter('gallahad')                 # a new counter from an iterable
>>> c = Counter({'a': 4, 'b': 2})           # a new counter from a mapping
>>> c = Counter(a=4, b=2)                   # a new counter from keyword args
```

一句话，Counter 是一个 dict，它的 value 是 key 在某个对象中出现的次数，Counter 对创建时，默认不赋值时为 0，另外，即使不存在，它的 key-value 应该是 obj-0，不是不存在。

2. elements()

```
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> c
Counter({'a': 4, 'b': 2, 'c': 0, 'd': -2})
>>>    c.elements()
<itertools.chain object at 0x000001B98469F9E8>
>>>    sorted(c.elements())
['a', 'a', 'a', 'a', 'b', 'b']
```

【注】Note, if an element's count has been set to zero or is a negative number, elements() will ignore it.

3. most_common(n=None)

返回一个列表，提供 n 个频率最高的元素和计数。如果没提供 n，或者是 None，most_common() 返回计数器中的所有元素。相等个数的元素顺序随机。

```
>>> Counter('abracadabra').most_common(3)
[('a', 5), ('b', 2), ('r', 2)]
>>> Counter('abracadabra').most_common()
[('a', 5), ('b', 2), ('r', 2), ('c', 1), ('d', 1)]
```