**9.6. collections module**

[**https://docs.python.org/2/library/collections.html#collections.Counter.elements**](https://docs.python.org/2/library/collections.html#collections.Counter.elements)

[**http://book.pythontips.com/en/latest/collections.html**](http://book.pythontips.com/en/latest/collections.html)

**https://www.pythonforbeginners.com/collection/python-collections-counter**

Dictionary is an **unordered set of key and value pair**. It is a container that contains data, enclosed within curly braces.

*New in version 2.4.*

**Source code:** [Lib/collections.py](https://github.com/python/cpython/tree/2.7/Lib/collections.py) and [Lib/\_abcoll.py](https://github.com/python/cpython/tree/2.7/Lib/_abcoll.py)

This module implements specialized container datatypes providing alternatives to Python’s general purpose built-in containers, **list**, [**tuple**](https://docs.python.org/2/library/functions.html#tuple), [**dict**](https://docs.python.org/2/library/stdtypes.html#dict), [**set**](https://docs.python.org/2/library/stdtypes.html#set)

|  |  |  |
| --- | --- | --- |
| [**namedtuple()**](https://docs.python.org/2/library/collections.html#collections.namedtuple) | factory function for creating tuple subclasses with named fields | *New in version 2.6.* |
| [**deque**](https://docs.python.org/2/library/collections.html#collections.deque) | list-like container with fast appends and pops on either end | *New in version 2.4.* |
| [**Counter**](https://docs.python.org/2/library/collections.html#collections.Counter) | dict subclass for counting hashable objects | *New in version 2.7.* |
| [**OrderedDict**](https://docs.python.org/2/library/collections.html#collections.OrderedDict) | dict subclass that remembers the order entries were added | *New in version 2.7.* |
| [**defaultdict**](https://docs.python.org/2/library/collections.html#collections.defaultdict) | dict subclass that calls a factory function to supply missing values | *New in version 2.5.* |

In addition to the concrete container classes, the collections module provides [**abstract base classes**](https://docs.python.org/2/library/collections.html#collections-abstract-base-classes) that can be used to test whether a class provides a particular interface, for example, whether it is **hashable** or a **mapping**.

## Counter :

## A counter tool is provided to support convenient and rapid tallies

## class collections.Counter([iterable-or-mapping])

from collections import Counter

count = Counter()

for word in [*'red'*, *'blue'*, *'red'*, *'green'*, *'blue'*, *'blue'*]:

count[word] = 1

print(*"Word count = "*, count)

count.clear()

print(“---------------------------------“)

for number in [1,2,3,4,3,2,1,5,4,3,2,1,1,5]:

count[number] = 1

## print(*"Number count = "*, count)

A [**Counter**](https://docs.python.org/2/library/collections.html#collections.Counter) is a **[dict](https://docs.python.org/2/library/stdtypes.html" \l "dict" \o "dict)** subclass for counting **hashable** objects. It is an unordered collection where elements are stored as dictionary keys and their counts are stored as dictionary values. Counts are allowed to be any integer value including zero or negative counts. The [Counter](https://docs.python.org/2/library/collections.html#collections.Counter) class is similar to bags or multisets in other languages.

Elements are counted from an ***iterable*** or initialized from another *mapping* (or counter):

**>>>** c = Counter() *# a new, empty counter*

**>>>** c = Counter('gallahad') *# a new counter from an iterable*

**>>>** c = Counter({'red': 4, 'blue': 2}) *# a new counter from a mapping*

**>>>** c = Counter(cats=4, dogs=8) *# a new counter from keyword args*

## Counter objects have a dictionary interface except that they return a zero count for missing items instead of raising a [KeyError](https://docs.python.org/2/library/exceptions.html" \l "exceptions.KeyError" \o "exceptions.KeyError):

**>>>** c = Counter(['eggs', 'ham'])

**>>>** c['bacon'] *# count of a missing element is zero*

0

## Setting a count to zero does not remove an element from a counter. Use del to remove it entirely:

**>>>** c['sausage'] = 0 *# counter entry with a zero count*

**>>> del** c['sausage'] *# del actually removes the entry*

## Counter objects support three methods beyond those available for all dictionaries:

## 1.elements() : Return an iterator over elements repeating each as many times as its count. Elements are returned in arbitrary order. If an element’s count is less than one, [elements()](https://docs.python.org/2/library/collections.html#collections.Counter.elements) will ignore it.

**>>>** c = Counter(a=4, b=2, c=0, d=-2)

**>>>** list(c.elements())

['a', 'a', 'a', 'a', 'b', 'b']

## 2.most\_common([*n*]):Return a list of the n most common elements and their counts from the most common to the least.

## If n is omitted or None, [most\_common()](https://docs.python.org/2/library/collections.html" \l "collections.Counter.most_common" \o "collections.Counter.most_common) returns all elements in the counter. Elements with equal counts are ordered arbitrarily:

**>>>** Counter('abracadabra').most\_common(3)

[('a', 5), ('r', 2), ('b', 2)]

**3. subtract(**[iterable-or-mapping]) **:** Elements are subtracted from an iterable or from another mapping (or counter). Like [dict.update()](https://docs.python.org/2/library/stdtypes.html" \l "dict.update" \o "dict.update) but subtracts counts instead of replacing them. Both inputs and outputs may be zero or negative.

**>>>** c = Counter(a=4, b=2, c=0, d=-2)

**>>>** d = Counter(a=1, b=2, c=3, d=4)

**>>>** c.subtract(d)

**>>>** c

Counter({'a': 3, 'b': 0, 'c': -3, 'd': -6})