

Counter — Count Hashable Objects

A Counter is a container that keeps track of how many times equivalent values are added. It can be used to implement the same algorithms for which other languages commonly use bag or multiset data structures.

Initializing

Counter supports three forms of initialization. Its constructor can be called with a sequence of items, a dictionary containing keys and counts, or using keyword arguments that map string names to counts.

```
# collections_counter_init.py

import collections

print(collections.Counter(['a', 'b', 'c', 'a', 'b', 'b']))
print(collections.Counter({'a': 2, 'b': 3, 'c': 1}))
print(collections.Counter(a=2, b=3, c=1))
```

The results of all three forms of initialization are the same.

```
$ python3 collections_counter_init.py

Counter({'b': 3, 'a': 2, 'c': 1})
Counter({'b': 3, 'a': 2, 'c': 1})
Counter({'b': 3, 'a': 2, 'c': 1})
```

An empty Counter can be constructed with no arguments and populated via the `update()` method.

```
# collections_counter_update.py

import collections

c = collections.Counter()
print('Initial :', c)

c.update('abcdaab')
print('Sequence:', c)

c.update({'a': 1, 'd': 5})
print('Dict      :', c)
```

The count values are increased based on the new data, rather than replaced. In the preceding example, the count for a goes from 3 to 4.

```
$ python3 collections_counter_update.py

Initial : Counter()
Sequence: Counter({'a': 3, 'b': 2, 'c': 1, 'd': 1})
Dict     : Counter({'d': 6, 'a': 4, 'b': 2, 'c': 1})
```

Accessing Counts

Once a Counter is populated, its values can be retrieved using the dictionary API.

```
# collections_counter_get_values.py

import collections

c = collections.Counter('abcdaab')
```

```
for letter in 'abcde':
    print('{} : {}'.format(letter, c[letter]))
```

Counter does not raise `KeyError` for unknown items. If a value has not been seen in the input (as with `e` in this example), its count is 0.

```
$ python3 collections_counter_get_values.py

a : 3
b : 2
c : 1
d : 1
e : 0
```

The `elements()` method returns an iterator that produces all of the items known to the Counter.

```
# collections_counter_elements.py

import collections

c = collections.Counter('extremely')
c['z'] = 0
print(c)
print(list(c.elements()))
```

The order of elements is not guaranteed, and items with counts less than or equal to zero are not included.

```
$ python3 collections_counter_elements.py

Counter({'e': 3, 'x': 1, 't': 1, 'r': 1, 'm': 1, 'l': 1, 'y': 1,
'z': 0})
['e', 'e', 'e', 'x', 't', 'r', 'm', 'l', 'y']
```

Use `most_common()` to produce a sequence of the n most frequently encountered input values and their respective counts.

```
# collections_counter_most_common.py

import collections

c = collections.Counter()
with open('/usr/share/dict/words', 'rt') as f:
    for line in f:
        c.update(line.rstrip().lower())

print('Most common:')
for letter, count in c.most_common(3):
    print('{}: {:>7}'.format(letter, count))
```

This example counts the letters appearing in all of the words in the system dictionary to produce a frequency distribution, then prints the three most common letters. Leaving out the argument to `most_common()` produces a list of all the items, in order of frequency.

```
$ python3 collections_counter_most_common.py

Most common:
e: 235331
i: 201032
a: 199554
```

Arithmetic

Counter instances support arithmetic and set operations for aggregating results. This example shows the standard operators for creating new Counter instances, but the in-place operators `+=`, `-=`, `&=`, and `|=` are also supported.

```
# collections_counter_arithmetic.py
```

import collections

```
c1 = collections.Counter(['a', 'b', 'c', 'a', 'b', 'b'])
c2 = collections.Counter('alphabet')

print('C1:', c1)
print('C2:', c2)

print('\nCombined counts:')
print(c1 + c2)

print('\nSubtraction:')
print(c1 - c2)

print('\nIntersection (taking positive minimums):')
print(c1 & c2)

print('\nUnion (taking maximums):')
print(c1 | c2)
```

Each time a new Counter is produced through an operation, any items with zero or negative counts are discarded. The count for a is the same in c1 and c2, so subtraction leaves it at zero.

```
$ python3 collections_counter_arithmetic.py
```

```
C1: Counter({'b': 3, 'a': 2, 'c': 1})
C2: Counter({'a': 2, 'l': 1, 'p': 1, 'h': 1, 'b': 1, 'e': 1, 't': 1})

Combined counts:
Counter({'a': 4, 'b': 4, 'c': 1, 'l': 1, 'p': 1, 'h': 1, 'e': 1, 't': 1})

Subtraction:
Counter({'b': 2, 'c': 1})

Intersection (taking positive minimums):
Counter({'a': 2, 'b': 1})

Union (taking maximums):
Counter({'b': 3, 'a': 2, 'c': 1, 'l': 1, 'p': 1, 'h': 1, 'e': 1, 't': 1})
```

Quick Links

- [Initializing](#)
- [Accessing Counts](#)
- [Arithmetic](#)

This page was last updated 2018-03-18.

Navigation

- [ChainMap — Search Multiple Dictionaries](#)
- [defaultdict — Missing Keys Return a Default Value](#)





[Get the book](#)

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for [examples for Python 2?](#)

This Site

-  [Module Index](#)
-  [Index](#)



© Copyright 2019, Doug Hellmann



Other Writing

-  [Blog](#)
-  [The Python Standard Library By Example](#)