Counting All Versions

Examples

Counting all occurence of all items in an iterable: collections.Counter

```
from collections import Counter

c = Counter(["a", "b", "c", "d", "a", "b", "a", "c", "d"])
c
# Out: Counter({'a': 3, 'b': 2, 'c': 2, 'd': 2})
c["a"]
# Out: 3

c[7]     # not in the list (7 occurred 0 times!)
# Out: 0
```

The collections.Counter can be used for any iterable and counts every occurrence for every element.

Note : One exception is if a dict or another collections.Mapping -like class is given, then it will not count them, rather it creates a Counter with these values:

```
Counter({"e": 2})
# Out: Counter({"e": 2})

Counter({"e": "e"})        # warning Counter does not verify the values are int
# Out: Counter({"e": "e"})
```

Counting the occurrences of one item in a sequence: list.count() and tuple.count()

```
alist = [1, 2, 3, 4, 1, 2, 1, 3, 4]
alist.count(1)
# Out: 3

atuple = ('bear', 'weasel', 'bear', 'frog')
atuple.count('bear')
# Out: 2
atuple.count('fox')
# Out: 0
```

Getting the most common value(-s): collections.Counter.most_common()

Counting the *keys* of a Mapping isn't possible with collections.Counter but we can count the *values* :

```
from collections import Counter
adict = {'a': 5, 'b': 3, 'c': 5, 'd': 2, 'e':2, 'q': 5}
Counter(adict.values())
# Out: Counter({2: 2, 3: 1, 5: 3})
```

The most common elements are avaiable by the most_common -method:

```
# Sorting them from most-common to least-common value:
Counter(adict.values()).most_common()
# Out: [(5, 3), (2, 2), (3, 1)]

# Getting the most common value
Counter(adict.values()).most_common(1)
# Out: [(5, 3)]

# Getting the two most common values
Counter(adict.values()).most_common(2)
# Out: [(5, 3), (2, 2)]
```

Counting occurences in numpy array

To count the occurences of a value in a numpy array. This will work:

```
>>> import numpy as np
>>> a=np.array([0,3,4,3,5,4,7])
>>> print np.sum(a==3)
2
```

The logic is that the boolean statement produces a array where all occurences of the requested values are 1 and all others are zero. So summing these gives the number of occurencies. This works for arrays of any shape or dtype.

There are two methods I use to count occurences of all unique values in numpy. Unique and bincount. Unique automatically flattens multidimensional arrays, while bincount only works with 1d arrays only containing positive integers.

```
>>> unique,counts=np.unique(a,return_counts=True)
>>> print unique,counts # counts[i] is equal to occurrences of unique[i] in a
[0 3 4 5 7] [1 2 2 1 1]
>>> bin_count=np.bincount(a)
>>> print bin_count # bin_count[i] is equal to occurrences of i in a
[1 0 0 2 2 1 0 1]
```

If your data are numpy arrays it is generally much faster to use numpy methods then to convert your data to generic methods.

Counting the occurrences of a substring in a string: str.count()

```
astring = 'thisisashorttext'
astring.count('t')
# Out: 4
```

This works even for substrings longer than one character:

```
astring.count('th')
# Out: 1
astring.count('is')
# Out: 2
astring.count('text')
# Out: 1
```

which would not be possible with `collections.Counter` which only counts single characters:

```
from collections import Counter
Counter(astring)
# Out: Counter({'a': 1, 'e': 1, 'h': 2, 'i': 2, 'o': 1, 'r': 1, 's': 3, 't': 4, 'x': 1})
```

Syntax

Parameters

Remarks