

ХОЧУ ПОМОЧЬ
ПРОЕКТУ

Класс Counter() модуля collections в Python

Ещё!

Распродажа

до 80%

Сроки проведения распродажи с 18.09.23 по 30.09.23. Товары, участвующие в распродаже, условия их распродажи определяются продавцами таких товаров и отмечаются значком «Распродажа». Представлен собирательный образ товара.

12+

avito.ru РЕКЛАМА · 16+

Больше информации на сайте рекламодателя

Подробнее

[Стандартная библиотека Python3.](#) / [Модуль collections в Python, контейнерные типы данных](#) / Класс Counter() модуля collections в Python

Подсчет количества повторений элементов в последовательности

[класс collections.Counter\(\)](#) предназначен для удобных и быстрых подсчетов количества появлений неизменяемых элементов в последовательностях.

```
>>> from collections import Counter
>>> cnt = Counter(['red', 'blue', 'red', 'green', 'blue', 'blue'])
>>> dict(cnt)
# {'blue': 3, 'red': 2, 'green': 1}
```

Синтаксис:

```
import collections
```

```
cnt = collections.Counter([iterable-or-mapping])
```

Параметры:

- iterable-or-mapping - итерируемая [последовательность](#) или [словарь](#).

Возвращаемое значение:

- объект Counter.

Описание:

Класс `Counter()` модуля `collections` - это подкласс `словаря dict` для подсчета хеш-объектов (неизменяемых, таких как `строки`, `числа`, `кортежи` и т.д.). Это коллекция, в которой элементы хранятся в виде словарных ключей, а их счетчики хранятся в виде значений словаря.

Счетчик может быть любым целочисленным значением, включая ноль или отрицательное число. Класс `collections.Counter()` похож на мультимножества в других языках программирования.

Элементы считаются из [итерируемой](#) последовательности, инициализируются из другого [словаря](#) или [счетчика Counter\(\)](#):

```
>>> from collections import Counter
# новый пустой счетчик
>>> cnt = Counter()
# новый счетчик из последовательности
>>> cnt = Counter('gallahad')
# новый счетчик из словаря
>>> cnt = Counter({'red': 4, 'blue': 2})
# новый счетчик из ключевых слов 'args'
>>> cnt = Counter(cats=4, dogs=8)
```

Счетчики `collections.Counter()` имеют интерфейс словаря, за исключением того, что они возвращают 0 для отсутствующих элементов вместо вызова исключения `KeyError`:

Вверх

```
>>> cnt = Counter(['eggs', 'ham'])
>>> cnt['bacon']
# 0
```

Установка счетчика в ноль не удаляет элементы из счетчика. Используйте [инструкцию del](#), чтобы полностью удалить ключ счетчика:

```
# запись счетчика с нулевым счетом
>>> cnt['sausage'] = 0
# удаление счетчика с нулевым счетом
>>> del cnt['sausage']
```

Смотрите больше [примеров использования класса Counter\(\)](#).

В качестве подкласса [dict\(\)](#), класс Counter() унаследовал возможность запоминания порядка вставки. Математические операции над объектами Counter() также сохраняют порядок. Результаты упорядочены в соответствии с тем, когда элемент сначала встречается в левом операнде, а затем в порядке, в котором встречается правый операнд.

Расширенные операторы сравнения.

Новое в версии 3.10: Добавлены расширенные операции сравнения.

С версии Python 3.10 счетчики Counter поддерживают расширенные [операторы сравнения](#) для отношений равенства, подмножества и надмножества: ==, !=, <, <=, >, >=. Все эти сравнения рассматривают отсутствующие элементы как имеющие нулевое количество, так что Counter(a=1) == Counter(a=1, b=0) возвращает True.

Изменено в версии 3.10: при сравнении на равенство, отсутствующие элементы рассматриваются как имеющие нулевое количество. Раньше Counter(a=3) и Counter(a=3, b=0) считались разными.

Объект Counter, атрибуты и методы:

- Counter.elements() [возвращает итератор по элементам](#).
- Counter.most_common() [список наиболее распространенных элементов](#).
- Counter.subtract() [вычитает элементы счетчика и итерируемой последовательности](#).
- Counter.total() [вычисляет сумму значений счетчика](#).
- Counter.update() [складывает элементы счетчика и итерируемой последовательности](#).

Counter.elements():

Метод [Counter.elements\(\)](#) возвращает итератор по элементам, в котором каждый элемент повторяется столько раз, во сколько установлено его значение. Элементы возвращаются в порядке их появления. Если количество элементов меньше единицы, то метод Counter.elements() просто проигнорирует его.

```
>>> from collections import Counter
>>> cnt = Counter(a=4, b=2, c=0, d=-2)
>>> sorted(cnt.elements())
# ['a', 'a', 'a', 'a', 'b', 'b']
```

Counter.most_common([n]):

Метод [Counter.most_common\(\)](#) возвращает список из n наиболее распространенных элементов и их количество от наиболее распространенных до наименее. Если n опущено или None, метод cnt.most_common() возвращает все элементы в счетчике.

Элементы с равным количеством упорядочены в порядке, в котором они встречаются первыми:

```
>>> from collections import Counter
>>> Counter('abracadabra').most_common(3)
# [('a', 5), ('b', 2), ('r', 2)]
```

Counter.subtract([iterable-or-mapping]):

Метод [Counter.subtract\(\)](#) вычитает элементы текущего счетчика cnt и итерируемой последовательности или другого [словаря](#) или другого [счетчика Counter\(\)](#). Подобно [методу словаря dict.update\(\)](#), но вычитает количество (значения ключей), а не заменяет их.

Значения ключей как у счетчика так и у словаря могут быть нулевыми или отрицательными.

```
>>> from collections import Counter
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> d = Counter(a=1, b=2, c=3, d=4)
>>> c.subtract(d)
>>> c
# { 'a': 3, 'b': 0, 'c': -3, 'd': -6 }
```

Counter.total():

В Python 3.10 появился метод [Counter.total\(\)](#), который вычисляет сумму значений текущего счетчика.

```
>>> from collections import Counter
>>> c = Counter(a=10, b=5, c=0)
>>> c.total()
# 15
```

В более ранних версиях Python, этот метод можно заменить выражением:

```
>>> from collections import Counter
>>> c = Counter(a=10, b=5, c=0)
>>> sum(c.values())
# 15
```

Для объектов collections.Counter() доступны обычные [методы словарей](#), за исключением двух, которые для счетчиков работают по-другому.

Counter.fromkeys(iterable):

Метод [Counter.fromkeys\(\)](#) не реализован для объектов [Counter\(\)](#).

Counter.update([iterable-or-mapping]):

Метод [Counter.update\(\)](#) складывает элементы текущего счетчика cnt и итерируемой последовательности или другого [словаря](#) или другого [счетчика Counter\(\)](#). Работает подобно [методу словаря dict.update\(\)](#), но складывает количество (значения ключей), а не заменяет их.

Кроме того, ожидается, что [итерация](#) будет [последовательностью](#) элементов, а не последовательностью двойных [кортежей](#) (key, value).

```
>>> from collections import Counter
>>> c = Counter(a=4, b=2, c=0, d=-2)
>>> d = Counter(a=1, b=2, c=3, d=4)
>>> c.update(d)
>>> c
# Counter({'a': 5, 'b': 4, 'c': 3, 'd': 2})
```

Примеры использования класса Counter().

Общие шаблоны для работы с объектами Counter:

```
>>> from collections import Counter
# создать счетчик из списка кортежей (elem, cnt)
>>> cnt = Counter(dict(list_of_pairs))
# преобразовать в список кортежей (elem, cnt)
>>> cnt.items()
# сумма всех значений в счетчике
>>> sum(cnt.values())
# очистить счетчик
>>> cnt.clear()
# список уникальных элементов
>>> list(cnt)
# преобразовать в множество
>>> set(cnt)
# преобразовать в обычный словарь
>>> dict(cnt)
# N наименее распространенных элементов
>>> cnt.most_common()[::-n-1:-1]
# Удалить элементы с нулевыми отрицательными значениями
>>> +cnt
```

Предусмотрено несколько математических операций для объединения объектов Counter() для создания мультимножеств (счетчиков, число которых больше нуля). Сложение и вычитание объединяют счетчики путем сложения или вычитания количества соответствующих элементов. Пересечение и объединение возвращают минимум и максимум соответствующих отсчетов.

Каждая операция принимает счетчики, значения ключей которых могут иметь любые числа, но выходные данные исключают из результата ключи с нулевыми отрицательными значениями.

Вверх

```
>> collections import Counter
>>> c = Counter(a=3, b=1)
```

```
>>> d = Counter(a=1, b=2)
# Сложить два счетчика:  c[x] + d[x]
>>> c + d
# Counter({'a': 4, 'b': 3})

# Вычитание счетчиков, сохраняются
# только положительные значения элементов
>>> c - d
# Counter({'a': 2})

# Пересечение счетчиков:  min(c[x], d[x])
>>> c & d
# Counter({'a': 1, 'b': 1})

# Объединение счетчиков:  max(c[x], d[x])
>>> c | d
# Counter({'a': 3, 'b': 2})
```

Унарное сложение и вычитание являются ссылками для добавления пустого счетчика или вычитания из пустого счетчика.

```
>>> from collections import Counter
>>> c = Counter(a=2, b=-4)
>>> +c
# Counter({'a': 2})
>>> -c
# Counter({'b': 4})
```

Подсчёт одинаковых слов в файле при помощи collections.Counter().

При помощи класса collections.Counter() можно легко подсчитать количество повторений слов в тексте. Например найдем десять самых распространенных слов в трагедия Уильяма Шекспира "Гамлет".

```
from collections import Counter
from re import findall

with open('hamlet.txt') as fp:
    words = findall(r'\w+', fp.read().lower())

cnt = Counter(words).most_common(10)
print(cnt)

# [('the', 1143), ('and', 966), ('to', 762),
#  ('of', 669), ('i', 631), ('you', 554),
#  ('a', 546), ('my', 514), ('hamlet', 471),
#  ('in', 451)]
```

Ограничения типа Counter() и минимальный диапазон:

Счетчики были в первую очередь предназначены для работы с положительными [целыми числами](#) для представления счетчиков.

Здесь описаны минимальный диапазон и ограничения типа [Counter\(\)](#), чтобы не исключать случаи использования, требующие хранения других типов или отрицательных значений.

- Сам [класс collections.Counter\(\)](#) является подклассом [словаря dict](#) без ограничений по ключам и значениям. Значения предназначены для чисел, представляющих счетчики, но вы можете хранить все что угодно в поле значений.
- Метод [cnt.most_common\(\)](#) только сортирует значения ключей по убыванию.
- Для операций на месте, таких как cnt[key] += 1, тип значения должен только поддерживать сложение и вычитание. Таким образом, дроби, числа с плавающей запятой и десятичные числа будут работать, отрицательные значения тоже поддерживаются. То же самое относится и к методам [cnt.update\(\)](#) и [cnt.subtract\(\)](#), которые допускают отрицательные и нулевые значения для входных и выходных данных.
- Мультимножественные методы предназначены только для случаев использования с положительными значениями. Входные значения ключей могут быть отрицательными или нулевыми, но в результате операций сохраняются только положительные значения. Нет ограничений по типу, но тип значения должен поддерживать сложение, вычитание и сравнение.
- Метод [Counter.elements\(\)](#) работает только с положительными [целыми числами](#) и будет игнорировать ноль и отрицательные значения.

Содержание раздела:

Вверх

[СТРАНИЦА РАЗДЕЛА](#)

- [Класс ChainMap\(\) модуля collections](#)
- [Класс deque\(\) модуля collections](#)
- [Класс Counter\(\) модуля collections](#)
- [Класс defaultdict\(\) модуля collections](#)
- РЕКЛАМА [Класс namedtuple\(\) модуля collections](#)
- [Класс OrderedDict\(\) модуля collections](#)
- [Класс UserString\(\) модуля collections](#)
- [Класс UserDict модуля collections](#)
- [Класс UserList модуля collections](#)