# Модуль collections

Модуль **collections** - предоставляет специализированные типы данных, на основе [словарей](https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html), [кортежей](https://pythonworld.ru/tipy-dannyx-v-python/kortezhi-tuple.html), [множеств](https://pythonworld.ru/tipy-dannyx-v-python/mnozhestva-set-i-frozenset.html), [списков](https://pythonworld.ru/tipy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html).

Первым рассматриваемым типом данных будет Counter.

## collections.Counter

**collections.Counter** - вид словаря, который позволяет нам считать количество неизменяемых объектов (в большинстве случаев, [строк](https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html)). Пример:

>>>

**>>> import** **collections**

**>>>** c = collections.Counter()

**>>> for** word **in** ['spam', 'egg', 'spam', 'counter', 'counter', 'counter']:

**...**  c[word] += 1

**...**

**>>> print**(c)

Counter({'counter': 3, 'spam': 2, 'egg': 1})

**>>> print**(c['counter'])

3

**>>> print**(c['collections'])

0

Но возможности Counter на этом не заканчиваются. У него есть несколько специальных методов:

**elements**() - возвращает список элементов в лексикографическом порядке.

>>>

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

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

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

**most\_common**([n]) - возвращает n наиболее часто встречающихся элементов, в порядке убывания встречаемости. Если n не указано, возвращаются все элементы.

>>>

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

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

**subtract**([iterable-or-mapping]) - вычитание

>>>

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

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

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

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

Наиболее часто употребляемые шаблоны для работы с Counter:

* sum(c.values()) - общее количество.
* c.clear() - очистить счётчик.
* list(c) - список уникальных элементов.
* set(c) - преобразовать в множество.
* dict(c) - преобразовать в словарь.
* c.most\_common()[:-n:-1] - n наименее часто встречающихся элементов.
* c += Counter() - удалить элементы, встречающиеся менее одного раза.

Counter также поддерживает сложение, вычитание, пересечение и объединение:

>>>

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

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

**>>>** c + d

Counter({'a': 4, 'b': 3})

**>>>** c - d

Counter({'a': 2})

**>>>** c & d

Counter({'a': 1, 'b': 1})

**>>>** c | d

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

Следующими на очереди у нас очереди (deque)

## collections.deque

**collections.deque**(iterable, [maxlen]) - создаёт очередь из итерируемого объекта с максимальной длиной maxlen. Очереди очень похожи на списки, за исключением того, что добавлять и удалять элементы можно либо справа, либо слева.

Методы, определённые в deque:

**append**(x) - добавляет x в конец.

**appendleft**(x) - добавляет x в начало.

**clear**() - очищает очередь.

**count**(x) - количество элементов, равных x.

**extend**(iterable) - добавляет в конец все элементы iterable.

**extendleft**(iterable) - добавляет в начало все элементы iterable (начиная с последнего элемента iterable).

**pop**() - удаляет и возвращает последний элемент очереди.

**popleft**() - удаляет и возвращает первый элемент очереди.

**remove**(value) - удаляет первое вхождение value.

**reverse**() - разворачивает очередь.

**rotate**(n) - последовательно переносит n элементов из начала в конец (если n отрицательно, то с конца в начало).

## collections.defaultdict

**collections.defaultdict** ничем не отличается от обычного словаря за исключением того, что по умолчанию всегда вызывается функция, возвращающая значение:

>>>

**>>> import** **collections**

**>>>** defdict = collections.defaultdict(list)

**>>> print**(defdict)

defaultdict(<class 'list'>, {})

**>>> for** i **in** range(5):

**...**  defdict[i].append(i)

**...**

**>>> print**(defdict)

defaultdict(<class 'list'>, {0: [0], 1: [1], 2: [2], 3: [3], 4: [4]})

## collections.OrderedDict

**collections.OrderedDict** - ещё один похожий на словарь объект, но он помнит порядок, в котором ему были даны ключи. Методы:

**popitem**(last=True) - удаляет последний элемент если last=True, и первый, если last=False.

**move\_to\_end**(key, last=True) - добавляет ключ в конец если last=True, и в начало, если last=False.

>>>

**>>>** d = {'banana': 3, 'apple':4, 'pear': 1, 'orange': 2}

**>>>** OrderedDict(sorted(d.items(), key=**lambda** t: t[0]))

OrderedDict([('apple', 4), ('banana', 3), ('orange', 2), ('pear', 1)])

**>>>** OrderedDict(sorted(d.items(), key=**lambda** t: t[1]))

OrderedDict([('pear', 1), ('orange', 2), ('banana', 3), ('apple', 4)])

**>>>** OrderedDict(sorted(d.items(), key=**lambda** t: len(t[0])))

OrderedDict([('pear', 1), ('apple', 4), ('orange', 2), ('banana', 3)])

## collections.namedtuple()

Класс **collections.namedtuple** позволяет создать тип данных, ведущий себя как кортеж, с тем дополнением, что каждому элементу присваивается имя, по которому можно в дальнейшем получать доступ:

>>>

**>>>** Point = namedtuple('Point', ['x', 'y'])

**>>>** p = Point(x=1, y=2)

**>>>** p

Point(x=1, y=2)

**>>>** p.x

1

**>>>** p[0]

1