



The collections module gives you ready-made containers for jobs that plain lists and dicts make awkward: counting, grouping, queues and small records. Import what you need: `from collections import Counter`.

COUNTER: COUNT THINGS

```
from collections import Counter
words = "the cat and the hat and the bat".split()
```

`Counter(words)`

→ `Counter({'the': 3, 'and': 2, 'cat': 1, 'hat': 1, 'bat': 1})`

a dict that maps each item to how many times it appears

`Counter(words)["the"]` → 3
read a count like a dict value

`Counter(words).most_common(2)`
→ `[('the', 3), ('and', 2)]`
the 2 most frequent items, biggest count first

`Counter("banana")`
→ `Counter({'a': 3, 'n': 2, 'b': 1})`
a string is counted letter by letter

`Counter(words).total()` → 8
3.10+: all the counts added up

COUNTER: ADD AND SUBTRACT

```
from collections import Counter
mon = Counter(tea=3, coffee=1)
tue = Counter(tea=1, cake=2)
```

`mon + tue`

→ `Counter({'tea': 4, 'cake': 2, 'coffee': 1})`

sales for both days together

`mon - tue`

→ `Counter({'tea': 2, 'coffee': 1})`
a count of 0 or less is dropped

`mon.update(["tea", "tea"])`
adds 2 to the tea count (dict.update would replace it)

`mon["tea"]` → 5

DEFAULTDICT: GROUP AND TOTAL

```
from collections import defaultdict
orders = [("Ada", 12), ("Linus", 5), ("Ada", 30)]
```

`by_customer = defaultdict(list)`
`for name, price in orders:`
 `by_customer[name].append(price)`

a missing key gets a starting value (here an empty list), so you can append straight away

`dict(by_customer)`
→ `{'Ada': [12, 30], 'Linus': [5]}`
orders grouped by customer

`spent = defaultdict(int)`
`for name, price in orders:`
 `spent[name] += price`
`int()` gives 0, so += works on a new key

`dict(spent)` → `{'Ada': 42, 'Linus': 5}`

`tags = defaultdict(set)`
`tags["Ada"].add("vip")`
`defaultdict(set)` collects values with no repeats

DEQUE: A QUEUE WITH TWO FAST ENDS

```
from collections import deque
q = deque(["Ada", "Grace"])
```

`q.append("Linus")` add to the right end

`q.appendleft("Hedy")` add to the left end

`q.popleft()` → 'Hedy'
remove from the left: first in, first out

`q.pop()` → 'Linus'
remove from the right

`q` → `deque(['Ada', 'Grace'])`

DEQUE: LAST N ITEMS, ROTATE

```
from collections import deque
```

`recent = deque(maxlen=3)`
`for s in ["tea", "jam", "figs", "milk"]:`
 `recent.append(s)`

with maxlen, a full deque drops the oldest item when you add one

`list(recent)` → `['jam', 'figs', 'milk']`
the 3 most recent searches

`d = deque([1, 2, 3, 4])`
`d.rotate(1)`
`d`
→ `deque([4, 1, 2, 3])`
`rotate(1)` moves each item one place right; `rotate(-1)` moves left

NAMEDTUPLE: A TUPLE WITH FIELD NAMES

```
from collections import namedtuple
Product = namedtuple("Product", ["name", "price"])
p = Product("tea", 4.5)
```

`p.price` → 4.5
read a field by name

`p[0]` → 'tea'
`p` is still a tuple: indexing and unpacking work

`p` → `Product(name='tea', price=4.5)`
`p` prints with its field names

`p._replace(price=5.0)`
→ `Product(name='tea', price=5.0)`
a changed copy; `p` stays the same

`p._asdict()`
→ `{'name': 'tea', 'price': 4.5}`
the fields as a dict, for JSON or printing

ORDEREDDICT

A plain dict keeps insertion order since Python 3.7, so you rarely need an `OrderedDict`. Its main use is `move_to_end`.

```
from collections import OrderedDict
pages = OrderedDict(home=1, shop=2, cart=3)
```

`pages.move_to_end("home")`
move a key to the end

`list(pages)` → `['shop', 'cart', 'home']`

`pages.move_to_end("cart", last=False)`
move a key to the front

`list(pages)` → `['cart', 'shop', 'home']`

CHAINMAP: SETTINGS WITH DEFAULTS

```
from collections import ChainMap
defaults = {"theme": "light", "lang": "en"}
user = {"theme": "dark"}
settings = ChainMap(user, defaults)
```

`settings["theme"]` → 'dark'
found in the first dict, user

`settings["lang"]` → 'en'
not in user, so it comes from defaults

`dict(settings)`
→ `{'theme': 'dark', 'lang': 'en'}`
both dicts seen as one; nothing is copied

`settings["lang"] = "fr"`
a change always goes into the first dict, user

`defaults`
→ `{'theme': 'light', 'lang': 'en'}`
so the defaults stay as they were

WATCH OUT

```
from collections import Counter
words = "the cat and the hat and the bat".split()
from collections import defaultdict
```

`Counter(words)["dog"]` → 0
a missing key gives 0, not a `KeyError`, so a typo goes unnoticed

`d = defaultdict(list)`
`print(d["Ada"])`
`len(d)`
→ 1

just reading `d[key]` creates the key: check with `'in'` or `d.get(key)`

`q[50_000]`
a deque is fast at the ends but slow in the middle: use a list for indexing

`p.price = 5.0`
`AttributeError: a namedtuple cannot change; use p._replace(price=5.0)`

`c = Counter(tea=1)`
`c.subtract(tea=3)`
`c`
→ `Counter({'tea': -2})`
`subtract()` keeps counts below 0; the `-` operator drops them