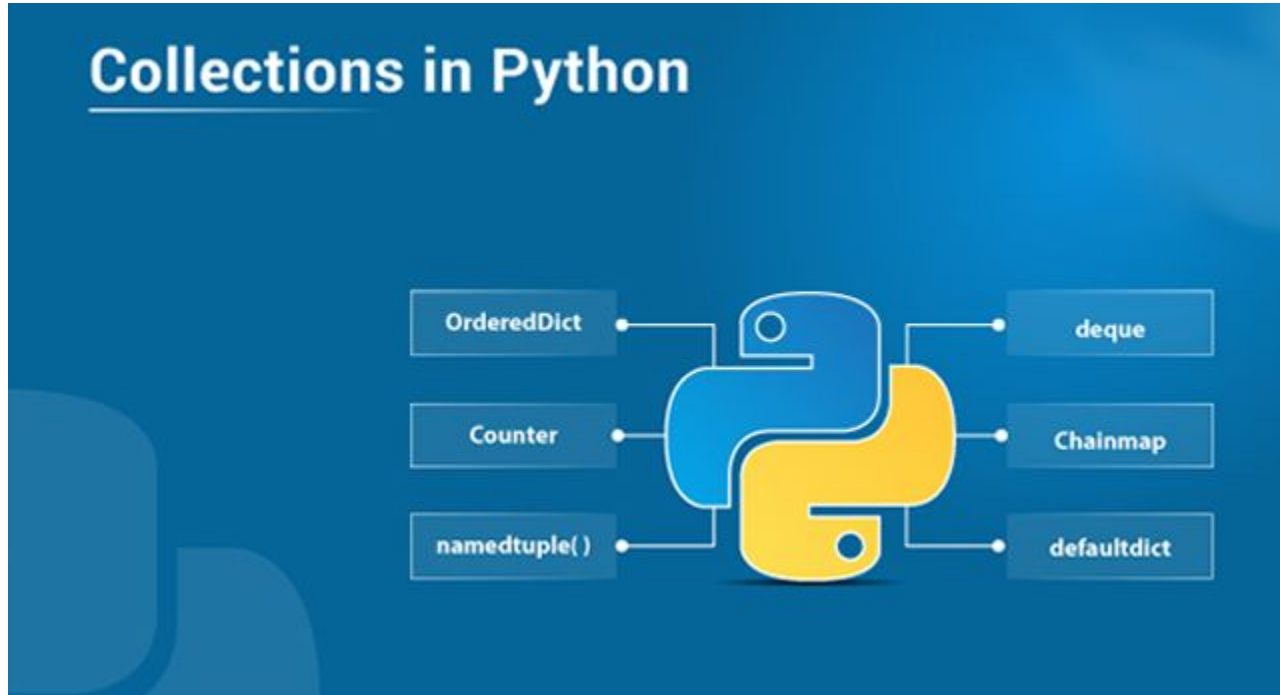


collections



collections overview

Python has a number of built-in collections which we use every day - lists, tuples, sets, dicts. In addition to them we have several useful data types in standard library

Today we are gonna to cover these ones

- Counter - useful when you need to ... well, count something
- defaultdict - nice in case of dicts with some objects as values

Counter

```
text = '''Lorem ipsum dolor sit amet, consectetur  
adipiscing elit, sed do eiusmod tempor incididunt ut  
labore et dolore magna aliqua. Ut enim ad minim  
veniam, quis nostrud exercitation ullamco laboris  
nisi ut aliquip ex ea commodo consequat. Duis aute  
irure dolor in reprehenderit in voluptate velit esse  
cillum dolore eu fugiat nulla pariatur. Excepteur  
sint occaecat cupidatat non proident, sunt in culpa  
qui officia deserunt mollit anim id est laborum. '''
```

```
from collections import Counter, defaultdict
```

```
Counter(text)
```

```
Counter({' ': 69, 'i': 42, 'e': 37, 't': 32, 'o': 29, 'a': 29, 'u': 28, 'n': 24, 'r': 22,  
'l': 21, 's': 18, 'd': 18, 'm': 17, 'c': 16, 'p': 11, 'q': 5, ',': 4, '.': 4, 'g': 3, 'b': 3,  
'v': 3, 'x': 3, 'f': 3, 'L': 1, 'U': 1, 'D': 1, 'h': 1, 'E': 1})
```

What's going on

As you can see, Counter applied to string counts frequency of each symbol

```
bag = ['pea', 'orange', 'pea', 'apple', 'apple',  
       'grape', 'mango', 'mango', 'mango']
```

```
Counter(bag)
```

```
Counter({'mango': 3, 'pea': 2, 'apple': 2, 'orange': 1, 'grape': 1})
```

This is generalized to every iterable with hashable elements (they can be dict keys) - elements of iterable are counted

Counter: You get counted
element: Counted by noob!

Counter methods

Counter is a variant of dictionary with additional methods

```
issubclass(Counter, dict)
```

True

- `most_common(n=None)` - returns list with tuples (element, count) sorted from most abundant to most rare; if `n` was specified returns list with tuples for `n` most abundant elements
- `elements()` - returns specific iterator, where you iterate count times for each corresponding element

```
numbers = (1, 2, 3, 1, 2, 1, 1, 10, 2, 1, 3, 4, 2, 5, 1, 7)
```

```
# Count it
```

```
freqs = Counter(numbers)
```

```
freqs
```

```
Counter({1: 6, 2: 4, 3: 2, 10: 1, 4: 1, 5: 1, 7: 1})
```

```
# Similar to list(freqs.items())
```

```
freqs.most_common()
```

```
[(1, 6), (2, 4), (3, 2), (10, 1), (4, 1), (5, 1), (7, 1)]
```

```
# Get most common element data
```

```
freqs.most_common(1)
```

```
[(1, 6)]
```

```
for elem in freqs.elements():  
    print(elem)
```

1

1

1

1

1

1

2

2

2

2

3

3

...

defaultdict

Really cool sometimes, because can free your code from unnecessary blocks of defining initial key values

```
orders = defaultdict(list)
```

```
orders
```

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

```
orders['today'].append('destroy enemies')
```

```
orders['today'].extend(('conquer the world', 'celebrate it'))
```

```
orders['today']
```

```
['destroy enemies', 'conquer the world', 'celebrate it']
```

orders

```
defaultdict(<class 'list'>, {'today': ['destroy enemies', 'conquer the world',  
                                         'celebrate it']})
```

Analogous to this block

```
orders2 = []
```

Create empty list as start point for this key

```
if 'today' not in orders2: # Also there is setdefault method  
    orders2['today'] = []
```

```
orders2['today'].append('destroy enemies')
```

```
orders2['today'].extend(('conquer the world', 'celebrate it'))
```

orders2

```
{'today': ['destroy enemies', 'conquer the world', 'celebrate it']}
```

Typical pipeline

1. Think do you need some default value for keys in your dict
2. If so, which value should it be?
 - a. you are going to store many things for 1 key - tuple, list, set or dict
 - b. you are going to operate with numbers - int or float
 - c. you wanna modify some complex object for each key - object of this class
3. Create defaultdict with selected default value
4. Operate with it in a proper manner
5. Profit!

```
# 1st example - store words and numbers in different keys  
text = '2430 AD is a good story, really. Read it if you  
haven\'t yet'
```

```
# Initial value will be []  
words = defaultdict(list)
```

```
for word in text.split():  
    if word.isdigit():  
        words['numbers'].append(word)  
    else:  
        words['words'].append(word)
```

```
words  
defaultdict(<class 'list'>, {'numbers': ['2430'], 'words':  
    ['AD', 'is', 'a', 'good', 'story,', 'really.', 'Read',  
    'it', 'if', 'you', 'haven't', 'yet']})
```

2nd example - I would like to store product of some values for 2 keys

```
some_values = [1, 2, 3, 3, 4, 5]
```

```
keys = cycle('AB')
```

Let initial value be 1

```
products = defaultdict(lambda: 1)
```

Simulate hard work

```
for key, number in zip(keys, some_values):  
    products[key] *= number
```

```
products
```

```
defaultdict(<function <lambda> at 0x7ff5e7e5d488>, {'A':  
12, 'B': 30})
```