deque — Double-Ended Queue

A double-ended queue, or deque, supports adding and removing elements from either end of the queue. The more commonly used stacks and queues are degenerate forms of deques, where the inputs and outputs are restricted to a single end.

```
# collections_deque.py

import collections

d = collections.deque('abcdefg')
print('Deque:', d)
print('Length:', len(d))
print('Left end:', d[0])
print('Right end:', d[-1])

d.remove('c')
print('remove(c):', d)
```

Since deques are a type of sequence container, they support some of the same operations as `list`, such as examining the contents with `__getitem__()`, determining length, and removing elements from the middle of the queue by matching identity.

```
$ python3 collections_deque.py

Deque: deque(['a', 'b', 'c', 'd', 'e', 'f', 'g'])
Length: 7
Left end: a
Right end: g
remove(c): deque(['a', 'b', 'd', 'e', 'f', 'g'])
```

Populating

A deque can be populated from either end, termed “left” and “right” in the Python implementation.

```
# collections_deque_populating.py

import collections

# Add to the right
d1 = collections.deque()
d1.extend('abcdefg')
print('extend    :', d1)
d1.append('h')
print('append    :', d1)

# Add to the left
d2 = collections.deque()
d2.extendleft(range(6))
print('extendleft:', d2)
d2.appendleft(6)
print('appendleft:', d2)
```

The `extendleft()` function iterates over its input and performs the equivalent of an `appendleft()` for each item. The end result is that the deque contains the input sequence in reverse order.

```
$ python3 collections_deque_populating.py

extend    : deque(['a', 'b', 'c', 'd', 'e', 'f', 'g'])
append    : deque(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
extendleft: deque([5, 4, 3, 2, 1, 0])
appendleft: deque([6, 5, 4, 3, 2, 1, 0])
```

Consuming

Similarly, the elements of the deque can be consumed from both ends or either end, depending on the algorithm being applied.

```python
# collections_deque_consuming.py

import collections

print('From the right:')
d = collections.deque('abcdefg')
while True:
    try:
        print(d.pop(), end='')
    except IndexError:
        break
print()

print('\nFrom the left:')
d = collections.deque(range(6))
while True:
    try:
        print(d.popleft(), end='')
    except IndexError:
        break
print()
```

Use `pop()` to remove an item from the “right” end of the deque and `popleft()` to take an item from the “left” end.

```
$ python3 collections_deque_consuming.py

From the right:
gfedcba

From the left:
012345
```

Since deques are thread-safe, the contents can even be consumed from both ends at the same time from separate threads.

```python
# collections_deque_both_ends.py

import collections
import threading
import time

candle = collections.deque(range(5))


def burn(direction, nextSource):
    while True:
        try:
            next = nextSource()
        except IndexError:
            break
        else:
            print('{:>8}: {}'.format(direction, next))
            time.sleep(0.1)
    print('{:>8} done'.format(direction))
    return


left = threading.Thread(target=burn,
                        args=('Left', candle.popleft))
right = threading.Thread(target=burn,
                         args=('Right', candle.pop))

left.start()
right.start()
```

```
left.join()
right.join()
```

The threads in this example alternate between each end, removing items until the deque is empty.

```
 $ python3 collections_deque_both_ends.py

 Left: 0
Right: 4
Right: 3
 Left: 1
Right: 2
 Left done
Right done
```

Rotating

Another useful aspect of the deque is the ability to rotate it in either direction, so as to skip over some items.

```
# collections_deque_rotate.py
```

```
import collections

d = collections.deque(range(10))
print('Normal        :', d)

d = collections.deque(range(10))
d.rotate(2)
print('Right rotation:', d)

d = collections.deque(range(10))
d.rotate(-2)
print('Left rotation :', d)
```

Rotating the deque to the right (using a positive rotation) takes items from the right end and moves them to the left end. Rotating to the left (with a negative value) takes items from the left end and moves them to the right end. It may help to visualize the items in the deque as being engraved along the edge of a dial.

```
$ python3 collections_deque_rotate.py

Normal        : deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Right rotation: deque([8, 9, 0, 1, 2, 3, 4, 5, 6, 7])
Left rotation : deque([2, 3, 4, 5, 6, 7, 8, 9, 0, 1])
```

Constraining the Queue Size

A deque instance can be configured with a maximum length so that it never grows beyond that size. When the queue reaches the specified length, existing items are discarded as new items are added. This behavior is useful for finding the last *n* items in a stream of undetermined length.

```
# collections_deque_maxlen.py
```

```
import collections
import random

# Set the random seed so we see the same output each time
# the script is run.
random.seed(1)

d1 = collections.deque(maxlen=3)
d2 = collections.deque(maxlen=3)

for i in range(5):
    n = random.randint(0, 100)
    print('n =', n)
    d1.append(n)
    d2.appendleft(n)
    print('D1:', d1)
    print('D2:', d2)
```

The deque length is maintained regardless of which end the items are added to.

```
$ python3 collections_deque_maxlen.py

n = 17
D1: deque([17], maxlen=3)
D2: deque([17], maxlen=3)
n = 72
D1: deque([17, 72], maxlen=3)
D2: deque([72, 17], maxlen=3)
n = 97
D1: deque([17, 72, 97], maxlen=3)
D2: deque([97, 72, 17], maxlen=3)
n = 8
D1: deque([72, 97, 8], maxlen=3)
D2: deque([8, 97, 72], maxlen=3)
n = 32
D1: deque([97, 8, 32], maxlen=3)
D2: deque([32, 8, 97], maxlen=3)
```

See also

- Wikipedia: Deque – A discussion of the deque data structure.
- Deque Recipes – Examples of using deques in algorithms from the standard library documentation.

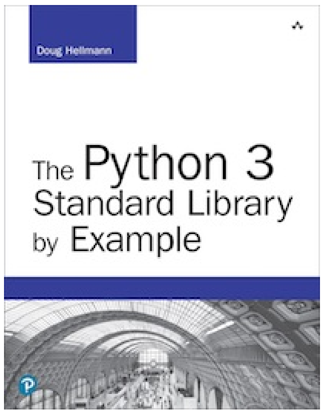
defaultdict — Missing Keys Return a Default Value   namedtuple — Tuple Subclass with Named Fields

Quick Links

This page was last updated 2017-07-30.

Navigation

defaultdict — Missing Keys Return a Default Value
namedtuple — Tuple Subclass with Named Fields



The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?