Specialised Data Types

Amit Kotlovski
Open-source consultancy
http://amitkot.com

Agenda

- Counter
- namedtuple
- defaultdict
- ordereddict
- deque

Counter

Given a paragraph, how can we count the number of times each character appears?

```python
text = """Do not dwell in the past,
do not dream of the future,
concentrate the mind on the present moment."""

input = text.strip()
occurrence = dict()
for c in input:
    try:
        occurrence[c] += 1
    except KeyError:
        occurrence[c] = 1

result = sorted(occurrence.items(),
                key=lambda i: i[1], reverse=True)
print(result)
```

```
[(' ', 16), ('t', 12), ('e', 12), ('n', 9), ('o', 8),
('r', 4), ('m', 4), ('h', 4), ('d', 4), ('a', 3),
('u', 2), ('s', 2), ('p', 2), ('l', 2), ('i', 2),
('f', 2), ('c', 2), (',', 2), ('\n', 2), ('w', 1),
('D', 1), ('.', 1)]
```

Using Counter

```
from collections import Counter

text = """Do not dwell in the past,
do not dream of the future,
concentrate the mind on the present moment."""

input = text.strip()
counted_occurence = Counter(input)
print(counted_occurence)
```

```
Counter({' ': 16, 'e': 12, 't': 12, 'n': 9, 'o': 8,
         'd': 4, 'h': 4, 'm': 4, 'r': 4, 'a': 3,
         '\n': 2, ',': 2, 'c': 2, 'f': 2, 'i': 2,
         'l': 2, 'p': 2, 's': 2, 'u': 2, '.': 1,
         'D': 1, 'w': 1})
```

Initialising

```python
from collections import Counter

# different ways to do the same
print(Counter(['first', 'second', 'last', 'second']))
print(Counter({'first': 1, 'second': 2, 'last': 1}))
print(Counter(first=1, second=2, last=1))
```

```
Counter({'second': 2, 'last': 1, 'first': 1})
Counter({'second': 2, 'last': 1, 'first': 1})
Counter({'second': 2, 'last': 1, 'first': 1})
```

Updating

```
from collections import Counter

>>> c = Counter()
>>> print(c)
Counter()

>>> c.update(a=10, b=10, c=10)
>>> print(c)
Counter({'a': 10, 'c': 10, 'b': 10})

>>> c.update(b=200, c=5)
>>> print(c)
Counter({'b': 210, 'c': 15, 'a': 10})
```

Accessing the counts

```
>>> from collections import Counter
>>> c = Counter('abbcccdddd')
>>> c['d']
4

>>> list(c.elements())
['b', 'b', 'c', 'c', 'c', 'd', 'd', 'd', 'd', 'a']

>>> c.most_common(2)
[('d', 4), ('c', 3)]
```

Counter operations

```
>>> c1 = Counter('abbcccdddd')
>>> c2 = Counter('ccddddee')
>>> c1 + c2
Counter({'d': 8, 'c': 5, 'b': 2, 'e': 2, 'a': 1})
>>> c1 - c2
Counter({'b': 2, 'c': 1, 'a': 1})
>>> c1 & c2
Counter({'d': 4, 'c': 2})
>>> c1 | c2
Counter({'d': 4, 'c': 3, 'b': 2, 'e': 2, 'a': 1})
```

Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

namedtuple

Let’s write a function for gathering data on shells

How do you return more than one type of data?

Source - /etc/passwd

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
.
.
.
```

Using tuples

```
from collections import Counter

def shell_data():
    with open('/etc/passwd') as f:
        shells = []
        for line in f:
           shells.append(line.strip().split(':')[-1])
        unique_shells = Counter(shells)
        num_of_users = len(shells)
        num_of_shells = len(unique_shells)
        name, count = unique_shells.most_common()[0]
        return num_of_users, num_of_shells, name, count

>>> print(shell_data())
(26, 5, '/bin/sh', 16)
```

tuples as return value

- tuples are nice
- But we lose context of the values passed
- Maybe define a value class instead?

```
from collections import Counter, namedtuple

ShellData = namedtuple("ShellData",
                       "unique_shells, num_of_users,
                        most_common, count”)
def shell_data():
    with open('/etc/passwd') as f:
        shells = []
        for line in f:
           shells.append(line.strip().split(':')[-1])
        unique_shells = Counter(shells)
        num_of_users = len(shells)
        num_of_shells = len(unique_shells)
        name, count = unique_shells.most_common()[0]
        return ShellData(num_of_users, num_of_shells,
                          name, count)

>>> print(shell_data())
ShellData(num_of_users=76, num_of_shells=3,
          most_common=‘/usr/bin/false', count=74)
```

Comparisons

```
>>> Car = namedtuple("Car", ["maker", "color", "size",
                              “license_plate_number"])

>>> c1 = Car("Ford", "Green", "big", "2341134")
>>> c2 = Car("Ford", "Green", "big", "2341134")
>>> c3 = Car("Susita", "Sand", "small", “4562391")

>>> c1 == c2
True
>>> c1 is c2
False
>>> c1 == c3
False
```

Reading from fields

```
>>> print(c1)
Car(maker='Ford', color='Green', size='big',
license_plate_number='2341134')

>>> print(c2.color)
Green

>>> c1.color = "Red"    # error - immutable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
```

Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

defaultdict

- A dictionary with default values

Let’s develop a system for processing orders in a restaurant

Gathering all orders into a list, and then feeding it into a dictionary of people and their orders

list -> dictionary

```python
order_data = [('Dan', 'Sushi'), ('Ariel', 'Lemonade'),
              ('Dan', 'Salad')]

orders = {}
for name, dish in order_data:
    try:
        orders[name].append(dish)
    except KeyError:
        orders[name] = [dish]

print orders
```

```
{'Ariel': ['Lemonade'], 'Dan': ['Sushi', 'Salad']}
```

- Manually detect keys not present
- Manually add new records

list -> defaultdict

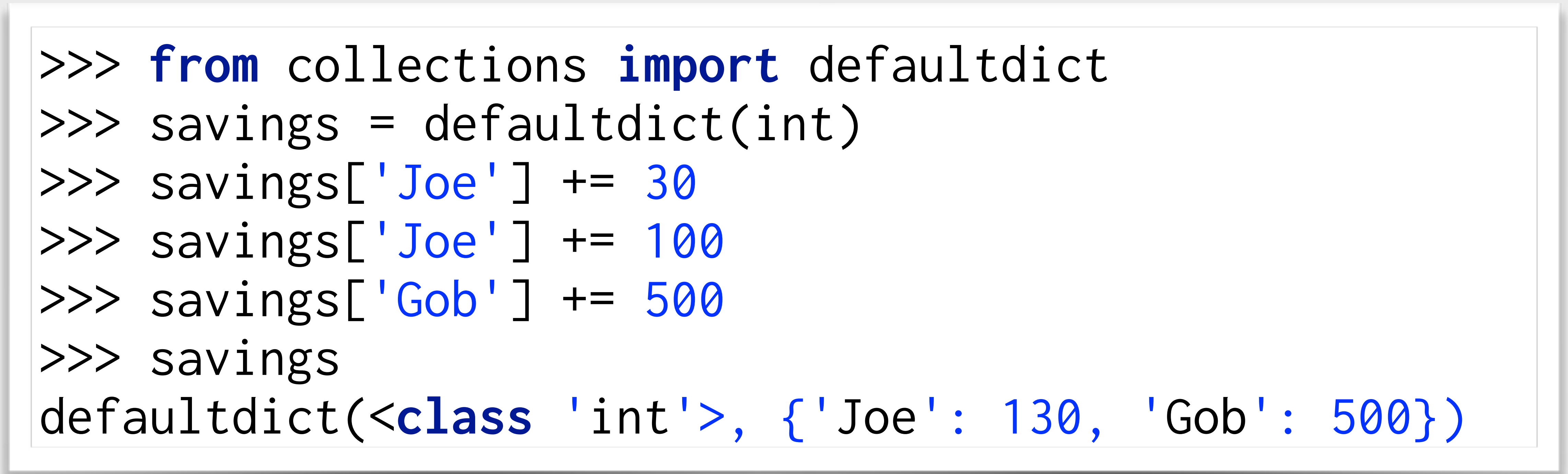
```python
from collections import defaultdict

order_data = [('Dan', 'Sushi'),
              ('Ariel', 'Lemonade'),
              ('Dan', 'Salad')]

orders_dict = defaultdict(list)
for name, item in order_data:
    orders_dict[name].append(item)
print(orders_dict)
```

```
(<type 'list'>, {'Ariel': ['Lemonade'],
                 'Dan': ['Sushi', 'Salad']})
```

defaultdict provides a default



```
>>> from collections import defaultdict
>>> savings = defaultdict(int)
>>> savings['Joe'] += 30
>>> savings['Joe'] += 100
>>> savings['Gob'] += 500
>>> savings
defaultdict(<class 'int'>, {'Joe': 130, 'Gob': 500})
```

Default string

```
>>> fruit_taste = defaultdict(lambda: 'Sweet')
>>> fruit_taste['grape'] = 'Sweet'
>>> fruit_taste.update(strawberry="Sweet",
                       Lemon="Sour",
                       Kiwi="Sweet+Sour")
>>> fruit_taste['Banana']
'Sweet'
>>> fruit_taste
defaultdict(<function <lambda> at 0x10e4936a8>,
{'Lemon': 'Sour', 'grape': 'Sweet', 'Banana': 'Sweet',
 'strawberry': 'Sweet', 'Kiwi': 'Sweet+Sour'})
```

Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

OrderedDict

- A dictionary that cares about order
- Items stored according to insertion order

Example

```
>>> from collections import OrderedDict
>>> d = OrderedDict()
>>> d['first'] = 1
>>> d['second'] = 2
>>> d['last'] = 3
>>> print(d)
OrderedDict([('first', 1), ('second', 2), ('last',
3)])

>>> d1 = dict()
>>> d1.update(first=1, second=2, last=3)
>>> print(d1)
{'second': 2, 'last': 3, 'first': 1}
```

Order counts - item iteration

```
>>> d1 = OrderedDict()
>>> d1['first'] = 1
>>> d1['second'] = 2
>>> d1['last'] = 3

>>> d2 = OrderedDict()
>>> d2['last'] = 3
>>> d2['second'] = 2
>>> d2['first'] = 1
```

```
>>> for k, v in d1.items():
...     print("{} -> {}".format(k, v))
...
first -> 1
second -> 2
last -> 3

>>> for k, v in d2.items():
...     print("{} -> {}".format(k, v))
...
last -> 3
second -> 2
first -> 1
```

Order counts - comparison

```
>>> d3 = OrderedDict()
>>> d3['last'] = 3
>>> d3['first'] = 1

>>> d2 = OrderedDict()
>>> d2['first'] = 1
>>> d2['last'] = 3

>>> d2
OrderedDict([('first', 1), ('last', 3)])
>>> d3
OrderedDict([('last', 3), ('first', 1)])
>>> d2 == d3    # same items, different order
False
```

More uses

```
>>> # regular unsorted dictionary
>>> d = {'banana': 3, 'apple':4, 'pear': 1,
         'orange': 2}

>>> # dictionary sorted by key *
>>> OrderedDict(sorted(d.items(), key=lambda t: t[0]))
OrderedDict([('apple', 4), ('banana', 3),
             ('orange', 2), ('pear', 1)])
```

* Later insertions will not maintain list sort key order

Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

deque

- Double ended queue
- Thread safe
- Memory efficient
- Support append and pop operations

pop and append act by default on the right side

```
>>> from collections import deque
>>> q = deque()
>>> q.append('first')
>>> q.append('second')
>>> q.append('last')
>>> q
deque(['first', 'second', 'last'])
>>> q.pop()
'last'
>>> q.pop()
'second'
>>> q
deque(['first'])
```

The left side

```
>>> q = deque('abcde')
>>> q
deque(['a', 'b', 'c', 'd', 'e'])
>>> q.popleft()
'a'
>>> q.popleft()
‘b'

>>> q.appendleft('START')
>>> q
deque(['START', 'c', 'd', 'e'])
```

- Lists are efficient when they have a fixed number of items
- Append and pop operations to the left side are O(n) in lists!
- `deque` is designed to support O(1) pop and append on both sides

Limiting deque length

New insertions push out old items

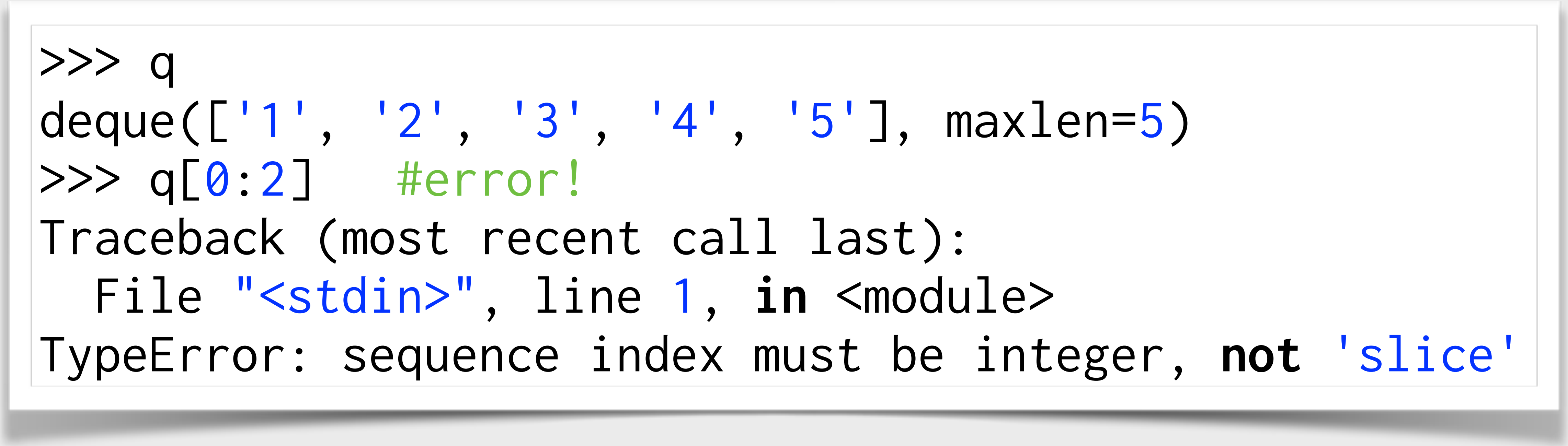
```
>>> q = deque('12345', maxlen=5)
>>> q
deque(['1', '2', '3', '4', '5'], maxlen=5)
>>> q.append(6)
>>> q
deque(['2', '3', '4', '5', 6], maxlen=5)
```

rotate

Circular shift of items inside the deque

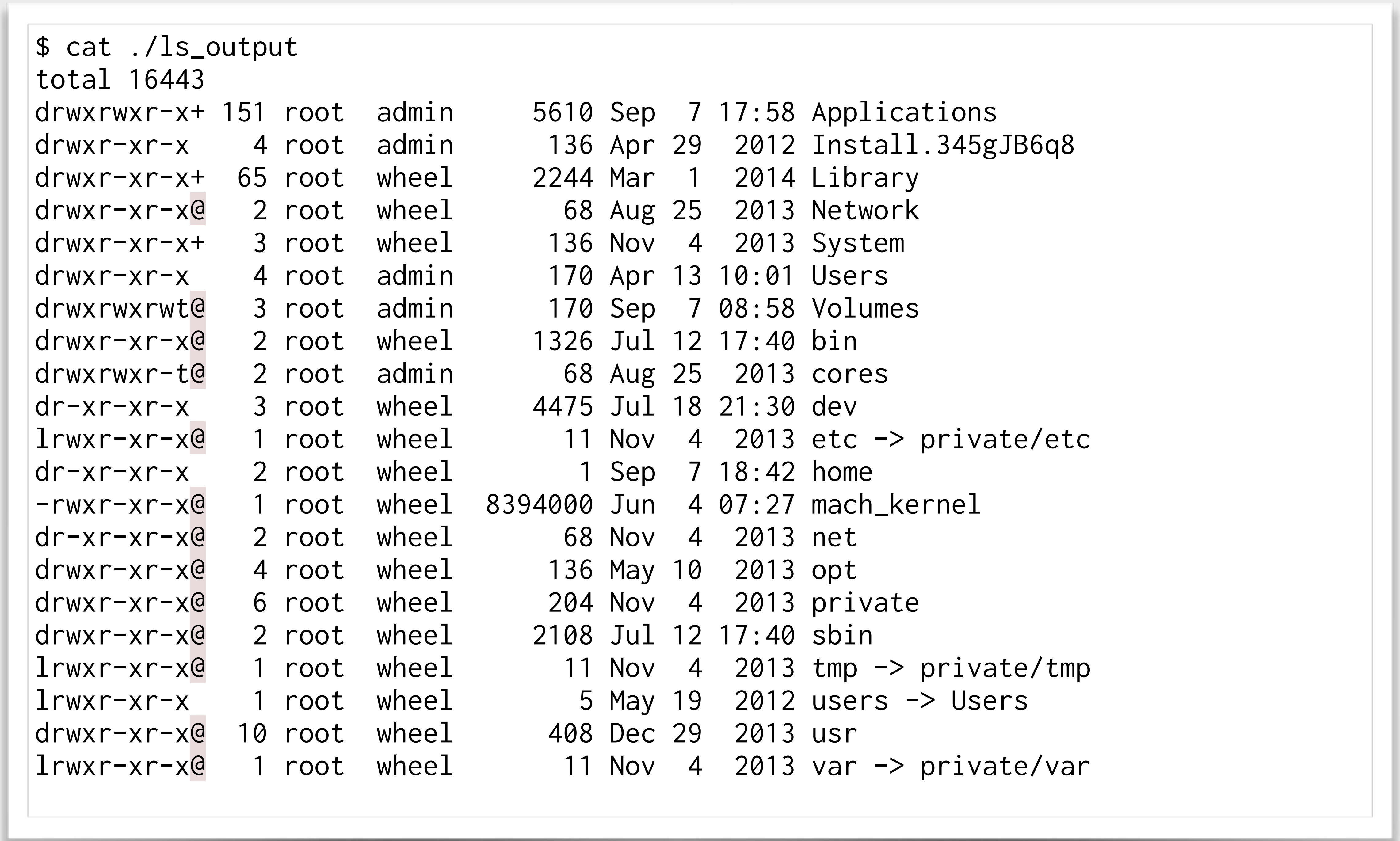
```
>>> q
deque(['1', '2', '3', '4', '5'], maxlen=5)
>>> q.rotate(2)
>>> q
deque(['4', '5', '1', '2', '3'], maxlen=5)
>>> q.rotate(-3)
>>> q
deque(['2', '3', '4', '5', '1'], maxlen=5)
```

Slicing is *not* supported



```
>>> q
deque(['1', '2', '3', '4', '5'], maxlen=5)
>>> q[0:2]   #error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence index must be integer, not 'slice'
```

Implementing Unix tail using deque



```
$ cat ./ls_output
total 16443
drwxrwxr-x+ 151 root  admin     5610 Sep  7 17:58 Applications
drwxr-xr-x    4 root  admin      136 Apr 29  2012 Install.345gJB6q8
drwxr-xr-x+  65 root  wheel     2244 Mar  1  2014 Library
drwxr-xr-x@   2 root  wheel       68 Aug 25  2013 Network
drwxr-xr-x+   3 root  wheel      136 Nov  4  2013 System
drwxr-xr-x    4 root  admin      170 Apr 13 10:01 Users
drwxrwxrwt@   3 root  admin      170 Sep  7 08:58 Volumes
drwxr-xr-x@   2 root  wheel     1326 Jul 12 17:40 bin
drwxrwxr-t@   2 root  admin       68 Aug 25  2013 cores
dr-xr-xr-x    3 root  wheel     4475 Jul 18 21:30 dev
lrwxr-xr-x@   1 root  wheel       11 Nov  4  2013 etc -> private/etc
dr-xr-xr-x    2 root  wheel        1 Sep  7 18:42 home
-rwxr-xr-x@   1 root  wheel  8394000 Jun  4 07:27 mach_kernel
dr-xr-xr-x@   2 root  wheel       68 Nov  4  2013 net
drwxr-xr-x@   4 root  wheel      136 May 10  2013 opt
drwxr-xr-x@   6 root  wheel      204 Nov  4  2013 private
drwxr-xr-x@   2 root  wheel     2108 Jul 12 17:40 sbin
lrwxr-xr-x@   1 root  wheel       11 Nov  4  2013 tmp -> private/tmp
lrwxr-xr-x    1 root  wheel        5 May 19  2012 users -> Users
drwxr-xr-x@  10 root  wheel      408 Dec 29  2013 usr
lrwxr-xr-x@   1 root  wheel       11 Nov  4  2013 var -> private/var
```

```python
from collections import deque

def tail(filename, lines=5):
    with open(filename) as f:
        return deque(f, maxlen=lines)

for line in tail('./ls_output'):
    print(line, end='')
```

```
drwxr-xr-x@   2 root   wheel     2108 Jul 12 17:40 sbin
lrwxr-xr-x@   1 root   wheel       11 Nov  4  2013 tmp -> private/tmp
lrwxr-xr-x    1 root   wheel        5 May 19  2012 users -> Users
drwxr-xr-x@  10 root   wheel      408 Dec 29  2013 usr
lrwxr-xr-x@   1 root   wheel       11 Nov  4  2013 var -> private/var
```

Q&A

https://secure.flickr.com/photos/21560098@N06/3836926854/

Summary

- We’ve seen useful specialised types in Python’s standard library
- There are other classes for specific uses, such as:
    - decimal
    - datetime’s date, time and datetime

AC..
QUESTION
EVERYTHING
TEAS EF
TLW

Thanks!

twitter	@amitkot
www	amitkot.com