30-collections

April 24, 2016

1 collections module

- extra data structures
- doc

2 defaultdict

- can give it a 'default function'
  - if a key doesn't exist, accessing it will create the key, and the value of the key will be the result of evaluting the default function
- avoids tedious checking for missing keys
- constructor takes an arg that creates missing keys

```
In [10]: # list is the default func
         # list() => []
         from collections import *

         dd = defaultdict(list)
         for n in '12394870984359872034598723 4':
             dd[n].append(n)
         dd
```

```
Out[10]: defaultdict(list,
                     {'0': ['0', '0'],
                      '1': ['1'],
                      '2': ['2', '2', '2'],
                      '3': ['3', '3', '3', '3'],
                      '4': ['4', '4', '4', '4'],
                      '5': ['5', '5'],
                      '7': ['7', '7', '7'],
                      '8': ['8', '8', '8', '8'],
                      '9': ['9', '9', '9', '9']})
```

```
In [7]: import random

        keys = []

        for j in range(10):
            keys.extend(j*[j])
        print(keys)

        # count incidence of keys
```

```
        dd = defaultdict(int)

        for k in keys:
            dd[k] += 1

        dd
```

```
[1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8
```

```
Out[7]: defaultdict(int, {1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9})
```

```
In [18]: # a lambda can supply non constant defaults
         # record access times

         import datetime

         dd = defaultdict(lambda : datetime.datetime.now().ctime())
         dd['zap']
         dd['zip']
         dd
```

```
Out[18]: defaultdict(<function __main__.<lambda>>,
                     {'zap': 'Thu Apr 21 11:08:42 2016',
                      'zip': 'Thu Apr 21 11:08:42 2016'})
```

3 deque

- 'double sided' list
- fast appends, pops, and extends on each end
- unlike list, has a rotate function
- doc

```
In [16]: d = deque(range(5))
         d
```

```
Out[16]: deque([0, 1, 2, 3, 4])
```

```
In [17]: # deque append is like list append

         d.append(100)
         d
```

```
Out[17]: deque([0, 1, 2, 3, 4, 100])
```

```
In [18]: # but can also append on the left side

         d.appendleft(200)
         d
```

```
Out[18]: deque([200, 0, 1, 2, 3, 4, 100])
```

```
In [19]: # likewise for pop

         print(d.pop())
         d
```

```
100
```

```
Out[19]: deque([200, 0, 1, 2, 3, 4])

In [20]: # pop on the left

         print(d.popleft())
         d

200

Out[20]: deque([0, 1, 2, 3, 4])

In [21]: d.rotate(2)
         d

Out[21]: deque([3, 4, 0, 1, 2])

In [22]: # usual element access

         [d[0],d[1], d[-1], d[-2]]

Out[22]: [3, 4, 2, 1]

In [25]: # maxlen can be useful when you only want to keep finite history

         d = deque(range(3), maxlen=3)
         d

Out[25]: deque([0, 1, 2])

In [26]: # 0 on the left is kicked out

         d.append(100)
         d

Out[26]: deque([1, 2, 100])

In [27]: # 100 on the right is kicked out

         d.appendleft(200)
         d

Out[27]: deque([200, 1, 2])
```

4 namedtuple

- associates field names with with tuple elements
- eliminate 'magic' index numbers

```
In [3]: xyzw = namedtuple('Point', 'x y z w')
        xyzw

Out[3]: __main__.Point

In [4]: # nice printing

        p = xyzw._make([1,2,3,4])
        p
```

```
Out[4]: Point(x=1, y=2, z=3, w=4)

In [5]: # can access by field name

        [p.x, p.y, p.z, p.w]

Out[5]: [1, 2, 3, 4]
```

5 heapq

- lives in its own module, not in collections

```
In [6]: from heapq import *
        import random
        ri = [ random.randint(0, 10) for j in range(40) ]
        ri

Out[6]: [9,
         9,
         5,
         8,
         2,
         10,
         7,
         0,
         1,
         7,
         1,
         9,
         8,
         10,
         5,
         8,
         4,
         6,
         8,
         2,
         1,
         5,
         3,
         10,
         6,
         0,
         0,
         1,
         9,
         3,
         7,
         4,
         8,
         10,
         6,
         8,
         3,
```

```
 9,
 2,
 7]
```

In [8]:
```
h = []
for j in ri:
    heappush(h, j)
h
```

Out[8]:
```
[0,
 1,
 0,
 2,
 1,
 0,
 1,
 4,
 2,
 1,
 3,
 8,
 5,
 7,
 3,
 4,
 6,
 5,
 3,
 7,
 2,
 7,
 5,
 10,
 10,
 9,
 6,
 10,
 9,
 9,
 7,
 9,
 8,
 10,
 8,
 8,
 6,
 9,
 8,
 8]
```

In [13]:
```
nlargest(4, h)
```

Out[13]:
```
[10, 10, 10, 9]
```

In [9]:
```
nsmallest(5,h)
```

Out[9]:
```
[0, 0, 0, 1, 1]
```