Python - Collections Module

The collections module provides alternatives to built-in container data types such as list, tuple and dict.

namedtuple()

The `namedtuple()` function returns a tuple-like object with named fields. These field attributes are accessible by lookup as well as by index.

General usage of this function is:

Signature:
`collections.namedtuple(type_name, field-list)`

The following statement declares a student class having name, age and marks as fields.

```
>>> import collections
>>> student=collections.namedtuple('student', [name, age,
marks])
```

To create a new object of this namedtuple, do the following:

```
>>> s1=student("Imran", 21, 98)
```

The values of the field can be accessible by attribute lookup:

```
>>> s1.name
'Imran'
```

Or by index:

```
>>>s1[0]
'Imran'
```

OrderedDict()

The `OrderedDict()` function is similar to a normal dictionary object in Python. However, it remembers the order of the keys in which they were first inserted.

```
import collections
d1=collections.OrderedDict()
d1['A']=65
d1['C']=67
d1['B']=66
d1['D']=68
```

```
for k,v in d1.items():
    print (k,v)
```

Output:

```
A 65
C 67
B 66
D 68
```

Upon traversing the dictionary, pairs will appear in the order of their insertion.

deque()

A deque object support appends and pops from either ends of a list. It is more memory efficient than a normal list object. In a normal list object, the removal of any item causes all items to the right to be shifted towards left by one index. Hence, it is very slow.

```
>>> q=collections.deque([10,20,30,40])
>>>q.appendleft(0)
>>>q
deque([0, 10, 20, 30, 40])
>>>q.append(50)
>>>q
deque([0, 10, 20, 30, 40, 50])
>>>q.pop()
50
>>>q
deque([0, 10, 20, 30, 40])
>>>q.popleft()
0
>>>q
deque([10, 20, 30, 40])
```