OrderedDict

Python's collections module has another great subclass of dict known as **OrderedDict**. As the name implies, this dictionary keeps track of the order of the keys as they are added. If you create a regular dict, you will note that it is an unordered data collection:

```
d = {'banana': 3, 'apple':4, 'pear': 1, 'orange': 2}
print (d)
#{'apple': 4, 'banana': 3, 'orange': 2, 'pear': 1}
```

Every time you print it out, the order may be different. There are times when you will need to loop over the keys of your dictionary in a specific order. For example, I have had a use case where I needed the keys sorted so I could loop over them in order. To do that, you can do the following:

```
keys = d.keys()
print (keys)
#dict_keys(['apple', 'orange', 'banana', 'pear'])

keys = sorted(keys)
print (keys)
#['apple', 'banana', 'orange', 'pear']

for key in keys:
    print (key, d[key])

#apple 4
#banana 3
#orange 2
#pear 1
```

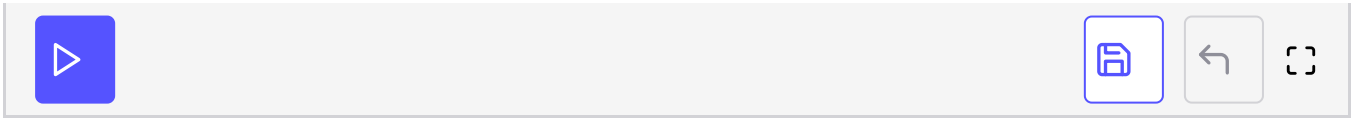
Let's create an instance of an OrderedDict using our original dict, but during

the creation, we'll sort the dictionary's keys:

```
from collections import OrderedDict
d = {'banana': 3, 'apple':4, 'pear': 1, 'orange': 2}
new_d = OrderedDict(sorted(d.items()))
print (new_d)
#OrderedDict([('apple', 4), ('banana', 3), ('orange', 2), ('pear', 1)])

for key in new_d:
    print (key, new_d[key])

#apple 4
#banana 3
#orange 2
#pear 1
```



Here we create our OrderedDict by sorting it on the fly using Python's sorted built-in function. The sorted function takes in the dictionary's items, which is a list of tuples that represent the key pairs of the dictionary. It sorts them and then passes them into the OrderedDict, which will retain their order. Thus when we go to print our the keys and values, they are in the order we expect. If you were to loop over a regular dictionary (not a sorted list of keys), the order would change all the time.

Note that if you add new keys, they will be added to the end of the OrderedDict instead of being automatically sorted.

Something else to note about OrderDicts is that when you go to compare two OrderedDicts, they will not only test the items for equality, but also that the order is correct. A regular dictionary only looks at the contents of the dictionary and doesn't care about its order.

Finally, OrderDicts have two new methods in Python 3: popitem and move_to_end. The popitem method will return and remove a (key, item) pair. The move_to_end method will move an existing key to either end of the OrderedDict. The item will be moved right end if the last argument for OrderedDict is set to True (which is the default), or to the beginning if it is False.

Interestingly, OrderedDicts support reverse iteration using Python's reversed built-in function:

```
for key in reversed(new_d):
    print (key, new_d[key])

#pear 1
#orange 2
#banana 3
#apple 4
```

Pretty neat, although you probably won't be needing that functionality every day.

Wrapping Up

We've covered a lot of ground in this chapter. You learned how to use a defaultdict and an OrderedDict. We also learned about a neat subclass of Python's list, the deque. Finally we looked at how to use a namedtuple to do various activities. I hope you found each of these collections interesting. They may be of great use to you in your own coding life.