Advanced Dictionaries

Unlike some of the other Data Structures we've worked with, most of the really useful methods available to us in Dictionaries have already been explored throughout this course. Here we will touch on just a few more for good measure:

Dictionary Comprehensions

Just like List Comprehensions, Dictionary Data Types also support their own version of comprehension for quick creation. It is not as commonly used as List Comprehensions, but the syntax is:

In [1]:
```
{x:x**2 for x in range(10)}
```

Out[1]:
```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

One of the reasons it is not as common is the difficulty in structuring key names that are not based off the values.

Iteration over keys, values, and items

Dictionaries can be iterated over using the keys(), values() and items() methods. For example:

In [2]:
```
d = {'k1':1,'k2':2}
```

In [3]:
```
for k in d.keys():
    print(k)
```

```
k1
k2
```

In [4]:
```
for v in d.values():
    print(v)
```

```
1
2
```

In [5]:
```
for item in d.items():
    print(item)
```

```
('k1', 1)
('k2', 2)
```

Viewing keys, values and items

By themselves the keys(), values() and items() methods return a dictionary *view object*. This is not a separate list of items. Instead, the view is always tied to the original dictionary.

In [6]:
```
key_view = d.keys()
```

```
key_view
```

Out[6]:
```
dict_keys(['k1', 'k2'])
```

In [7]:
```
d['k3'] = 3

d
```

Out[7]:
```
{'k1': 1, 'k2': 2, 'k3': 3}
```

In [8]:
```
key_view
```

Out[8]:
```
dict_keys(['k1', 'k2', 'k3'])
```

Great! You should now feel very comfortable using the variety of methods available to you in Dictionaries!