Dictionaries

We've been learning about *sequences* in Python but now we're going to switch gears and learn about *mappings* in Python. If you're familiar with other languages you can think of these Dictionaries as hash tables.

This section will serve as a brief introduction to dictionaries and consist of:

```
1.) Constructing a Dictionary
2.) Accessing objects from a dictionary
3.) Nesting Dictionaries
4.) Basic Dictionary Methods
```

So what are mappings? Mappings are a collection of objects that are stored by a *key*, unlike a sequence that stored objects by their relative position. This is an important distinction, since mappings won't retain order since they have objects defined by a key.

A Python dictionary consists of a key and then an associated value. That value can be almost any Python object.

Constructing a Dictionary

Let's see how we can construct dictionaries to get a better understanding of how they work!

```python
In [1]:  # Make a dictionary with {} and : to signify a key and a value
         my_dict = {'key1':'value1','key2':'value2'}
```

```python
In [2]:  # Call values by their key
         my_dict['key2']
```

```
Out[2]:  'value2'
```

Its important to note that dictionaries are very flexible in the data types they can hold. For example:

```python
In [3]:  my_dict = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}
```

```python
In [4]:  # Let's call items from the dictionary
         my_dict['key3']
```

```
Out[4]:  ['item0', 'item1', 'item2']
```

```python
In [5]:  # Can call an index on that value
         my_dict['key3'][0]
```

```
Out[5]:  'item0'
```

```
In [6]:  # Can then even call methods on that value
         my_dict['key3'][0].upper()
```

```
Out[6]:  'ITEM0'
```

We can affect the values of a key as well. For instance:

```
In [7]:  my_dict['key1']
```

```
Out[7]:  123
```

```
In [8]:  # Subtract 123 from the value
         my_dict['key1'] = my_dict['key1'] - 123
```

```
In [9]:  #Check
         my_dict['key1']
```

```
Out[9]:  0
```

A quick note, Python has a built-in method of doing a self subtraction or addition (or multiplication or division). We could have also used += or -= for the above statement. For example:

```
In [10]:  # Set the object equal to itself minus 123
          my_dict['key1'] -= 123
          my_dict['key1']
```

```
Out[10]:  -123
```

We can also create keys by assignment. For instance if we started off with an empty dictionary, we could continually add to it:

```
In [11]:  # Create a new dictionary
          d = {}
```

```
In [12]:  # Create a new key through assignment
          d['animal'] = 'Dog'
```

```
In [13]:  # Can do this with any object
          d['answer'] = 42
```

```
In [14]:  #Show
          d
```

```
Out[14]:  {'animal': 'Dog', 'answer': 42}
```

Nesting with Dictionaries

Hopefully you're starting to see how powerful Python is with its flexibility of nesting objects and calling methods on them. Let's see a dictionary nested inside a dictionary:

```python
In [15]:  # Dictionary nested inside a dictionary nested inside a dictionary
          d = {'key1':{'nestkey':{'subnestkey':'value'}}}
```

Wow! That's a quite the inception of dictionaries! Let's see how we can grab that value:

```python
In [16]:  # Keep calling the keys
          d['key1']['nestkey']['subnestkey']
```

```
Out[16]:  'value'
```

A few Dictionary Methods

There are a few methods we can call on a dictionary. Let's get a quick introduction to a few of them:

```python
In [17]:  # Create a typical dictionary
          d = {'key1':1,'key2':2,'key3':3}
```

```python
In [18]:  # Method to return a list of all keys
          d.keys()
```

```
Out[18]:  dict_keys(['key1', 'key2', 'key3'])
```

```python
In [19]:  # Method to grab all values
          d.values()
```

```
Out[19]:  dict_values([1, 2, 3])
```

```python
In [20]:  # Method to return tuples of all items  (we'll learn about tuples soon)
          d.items()
```

```
Out[20]:  dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```

Hopefully you now have a good basic understanding how to construct dictionaries. There's a lot more to go into here, but we will revisit dictionaries at later time. After this section all you need to know is how to create a dictionary and how to retrieve values from it.