Python course materials

# 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!

# Make a dictionary with {} and : to signify a key and a value  
my\_dict = {'key1':'value1','key2':'value2'}

# Call values by their key  
my\_dict['key2']

'value2'

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

my\_dict = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}

# Let's call items from the dictionary  
my\_dict['key3']

['item0', 'item1', 'item2']

# Can call an index on that value  
my\_dict['key3'][0]

'item0'

# Can then even call methods on that value  
my\_dict['key3'][0].upper()

'ITEM0'

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

my\_dict['key1']

123

# Subtract 123 from the value  
my\_dict['key1'] = my\_dict['key1'] - 123

#Check  
my\_dict['key1']

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:

# Set the object equal to itself minus 123   
my\_dict['key1'] -= 123  
my\_dict['key1']

-123

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

# Create a new dictionary  
d = {}

# Create a new key through assignment  
d['animal'] = 'Dog'

# Can do this with any object  
d['answer'] = 42

#Show  
d

{'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:

# 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:

# Keep calling the keys  
d['key1']['nestkey']['subnestkey']

'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:

# Create a typical dictionary  
d = {'key1':1,'key2':2,'key3':3}

# Method to return a list of all keys   
d.keys()

dict\_keys(['key1', 'key2', 'key3'])

# Method to grab all values  
d.values()

dict\_values([1, 2, 3])

# Method to return tuples of all items (we'll learn about tuples soon)  
d.items()

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.