Content Discussed in class_8_16-August-2016

- Chapter 4: Collections
    - Dictionaries
        - Two methods of creating a dictionary
        - Indexing the dictionaries
        - Editing an existing dictionary
        - Conditions
        - iterations on dictionaries
        - String Formatting with dictionaries
        - dictonary keys should be immutable
        - COPY in dictionaries
        - Creating dictionaries from lists
        - Memoization
        - Ordered Dictionary

Interview Questions Discussed

Interview Question 1: How to create a list of tuples?

Interview Question 2: What is memoization. How to achieve it in dictionaries?

Interview Question 3: How to sort a dictionary based on the length of the key?

Asssignments Given

Assignment 1: Explore the usage of these dictionaries attributes: 'pop', 'popitem'

Assignment 2: Write a script to take the names of our five friends in a list, and their designation s in a separate list. ... Then, create a dictionary, containing their name, designation pairs

Assignment 3: Write a script to get the letter frequency from a given sentence. The first three letters are repeated.

Assignment 4: Use collections.OrdererDict to get an ordered dictionary. Try to do some example

---

Dictionaries

- It is a key/value structure
- It contain series of key:value pair, separated by comma(,) operator and enclosed in {} .
- eg: dict1 = {key1:value1, key2: value2}
- It doesn't store duplicate keys
- The keys in dictionaries should be immutable (string, tuple, int, float, frozenset). whereas list, set is not possible.
- Indexing is done based on keys, and not position.

```
In [3]: d = {}  # Empty dictionary
```

```
In [4]: type(d), len(d), d
```

```
Out[4]: (dict, 0, {})
```

```
In [5]: d = {'a': 'apple', 'b': 'banana', 'c': 'cat'}   # method 1 of dictionary creat
        ion
```

```
In [6]: print d    # Observe the order of the elements
```

```
{'a': 'apple', 'c': 'cat', 'b': 'banana'}
```

```
In [7]: dict1 = {}
        dict1['A'] = 'Apple'
        dict1['B'] = 'Banana'
        dict1['B'] = 'Ball'
        dict1['c'] = 'Cat'
```

```
In [8]: print dict1         # updates the latest key, when removing duplicates
```

```
{'A': 'Apple', 'c': 'Cat', 'B': 'Ball'}
```

Indexing the dictionaries

```
In [9]: print "The is  one ", dict1['c'], " in the hall"
```

```
The is  one  Cat  in the hall
```

```
In [10]: print dict1['C']   # cae sensitivity
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-10-57cb1734cad6> in <module>()
----> 1 print dict1['C']

KeyError: 'C'
```

```
In [11]: 'C' in dict1  # membership check

Out[11]: False

In [12]: 'c' in dict1

Out[12]: True

In [13]: dict1.values()

Out[13]: ['Apple', 'Cat', 'Ball']

In [14]: dict1.keys()

Out[14]: ['A', 'c', 'B']

In [15]: dict1.items()

Out[15]: [('A', 'Apple'), ('c', 'Cat'), ('B', 'Ball')]
```

Editing an existing dictionary

```
In [16]: dict1['ac'] = 'Air Conditioner'

In [18]: dict1

Out[18]: {'A': 'Apple', 'B': 'Ball', 'ac': 'Air Conditioner', 'c': 'Cat'}

In [19]: dict1['z'] = 'Zombie'

In [20]: dict1

Out[20]: {'A': 'Apple', 'B': 'Ball', 'ac': 'Air Conditioner', 'c': 'Cat', 'z': 'Zombi
         e'}
```

Conditions

```
In [21]: if 'ac' in dict1:
             print 'There is ', dict1['ac']

         There is  Air Conditioner

In [22]: dict1.get('B')

Out[22]: 'Ball'

In [23]: dict1.get('b')  # Doesn't raise any error, in the absence of specified key
In [24]: dict1.get('b', 'xxx')  # if not present, returns the specified value

Out[24]: 'xxx'
```

iterations on dictionaries

```
In [25]: [i for i in dict1.items()]

Out[25]: [('A', 'Apple'),
          ('c', 'Cat'),
          ('B', 'Ball'),
          ('ac', 'Air Conditioner'),
          ('z', 'Zombie')]
```

```
In [26]: [i for i in dict1]

Out[26]: ['A', 'c', 'B', 'ac', 'z']
```

NOTE: By default, iterating over a dictionary takes place on keys() only

```
In [27]: [i for i in dict1.keys()]

Out[27]: ['A', 'c', 'B', 'ac', 'z']
```

```
In [28]: [i for i in dict1.values()]

Out[28]: ['Apple', 'Cat', 'Ball', 'Air Conditioner', 'Zombie']
```

String Formatting with dictionaries

```
In [29]: cricket ={'players': 'Batsmen and Bowlers', 'count': 11}
```

```
In [30]: cricket

Out[30]: {'count': 11, 'players': 'Batsmen and Bowlers'}
```

```
In [32]: sentence = "The %(players)s in cricket team are %(count)d in number!"%cricket
```

```
In [33]: sentence

Out[33]: 'The Batsmen and Bowlers in cricket team are 11 in number!'
```

```
In [34]: sentence = "The %(players)r in cricket team are %(count)r in number!"%cricket
```

```
In [35]: sentence

Out[35]: "The 'Batsmen and Bowlers' in cricket team are 11 in number!"
```

In [36]:
```
sentence = "The %(0)s in cricket team are %(1)d in number!"%
(cricket['players'], cricket['count'])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-4332fc0d72a8> in <module>()
----> 1 sentence = "The %(0)s in cricket team are %(1)d in number!"%
(cricket['players'], cricket['count'])

TypeError: format requires a mapping
```

In [37]:
```
dict1
```

Out[37]:
```
{'A': 'Apple', 'B': 'Ball', 'ac': 'Air Conditioner', 'c': 'Cat', 'z': 'Zombi
e'}
```

In [39]:
```
dict1.clear()   # results in empty dictionary object
```

In [40]:
```
dict1
```

Out[40]:
```
{}
```

In [41]:
```
del dict1      # deletes the object
```

In [42]:
```
dict1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-42-8239e7494a4a> in <module>()
----> 1 dict1

NameError: name 'dict1' is not defined
```

dictonary keys should be immutable

In [44]:
```
dict1 = {[1,2,3]: 'numbers'} # Not possible, as list is mutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-62edb3b9df46> in <module>()
----> 1 dict1 = {[1,2,3]: 'numbers'}

TypeError: unhashable type: 'list'
```

In [45]:
```
dict1 = {(1,2,3): 'numbers'}  #possible, as tuple is immutable
```

In [46]:
```
dict1 = {"1,2,3": 'numbers'}  # possible, as string is immutable
```

In [47]:
```
dict1 = {123: 'numbers'}  # possible, as int is immutable
```

```
In [48]: dict1

Out[48]: {123: 'numbers'}
```

```
In [49]: dict1 = {{1,2,3}: 'numbers'}  # Not possible, as set is mutable

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-7755ecdb2543> in <module>()
----> 1 dict1 = {{1,2,3}: 'numbers'}  # Not possible, as set is mutable

TypeError: unhashable type: 'set'
```

```
In [50]: dict1 = {frozenset({1,2,3}): 'numbers'}  #  possible, as frozen set is immutab
         le
```

```
In [51]: dict1

Out[51]: {frozenset({1, 2, 3}): 'numbers'}
```

```
In [52]: dict1 = {3: 'numbers'}  # on integers
```

```
In [53]: dict1 = {3.333: 'numbers'}  # on floats
```

```
In [54]: dict1 = {True: 'numbers'}  # on booleans
```

```
In [58]: dict1 = {True: 'numbers', 1: 'one', 2: 'two', 3: 'three'}
```

```
In [62]: dict1

Out[62]: {True: 'one', 2: 'two', 3: 'three'}
```

As 'True' is a python object, it is preferred.

COPY in dictionaries

```
In [59]: dictHardCopy = dict1 # Hard COPY
```

```
In [60]: dictCopy = dict1.copy()  # soft COPY
```

```
In [61]: print dict1, '\n', dictHardCopy, '\n', dictCopy

         {True: 'one', 2: 'two', 3: 'three'}
         {True: 'one', 2: 'two', 3: 'three'}
         {True: 'one', 2: 'two', 3: 'three'}
```

In [63]:
```
dict1 == dictHardCopy == dictCopy
```
Out[63]:
```
True
```

In [64]:
```
dict1 is dictHardCopy is dictCopy
```
Out[64]:
```
False
```

In [65]:
```
dictHardCopy[3] = '3333333'  # updating the key, not position
```

In [66]:
```
dictCopy[2] = '2222222'  # indexing is done with key, and not index here
```

In [67]:
```
print dict1, '\n', dictHardCopy, '\n', dictCopy
```
```
{True: 'one', 2: 'two', 3: '3333333'}
{True: 'one', 2: 'two', 3: '3333333'}
{True: 'one', 2: '2222222', 3: 'three'}
```

In [68]:
```
dict2 = {'a': 'apple', True: 'one', 2: '2222', 3: 'three', 'b': 'ball'}
```

In [69]:
```
sorted(dict2)
```
Out[69]:
```
[True, 2, 3, 'a', 'b']
```

In [70]:
```
sorted(dict2.values())
```
Out[70]:
```
['2222', 'apple', 'ball', 'one', 'three']
```

In [71]:
```
sorted(dict2.items())
```
Out[71]:
```
[(True, 'one'), (2, '2222'), (3, 'three'), ('a', 'apple'), ('b', 'ball')]
```

In [72]:
```
dict2
```
Out[72]:
```
{True: 'one', 2: '2222', 3: 'three', 'a': 'apple', 'b': 'ball'}
```

In [73]:
```
d1 = {'a': 'apple', 'b': 'banana'}
```

In [74]:
```
d2 = {'a': 'america', 'b': 'bahamas', 'c':'canada'}  # observe the common keys
 in d1 and d2
```

In [75]:
```
print d1
```
```
{'a': 'apple', 'b': 'banana'}
```

In [76]:
```
print d2
```
```
{'a': 'america', 'c': 'canada', 'b': 'bahamas'}
```

In [77]:
```
d2.update(d1)
```

```
In [78]: d2
Out[78]: {'a': 'apple', 'b': 'banana', 'c': 'canada'}

In [79]: d1.update(d2)

In [80]: d1
Out[80]: {'a': 'apple', 'b': 'banana', 'c': 'canada'}
```

creating dictionaries from lists

```
In [82]: countries = ['India', 'US', 'UK', 'Germany']

In [83]: capitals = ['New Delhi', 'Washington', 'London', 'Berlin']

In [84]: countriesNcaptitals = zip(countries, capitals)

In [85]: countriesNcaptitals
Out[85]: [('India', 'New Delhi'),
          ('US', 'Washington'),
          ('UK', 'London'),
          ('Germany', 'Berlin')]
```

Interview Question 1: How to create a list of tuples?

Ans: Using zip, map

```
In [86]: type(countriesNcaptitals)
Out[86]: list

In [87]: cncDictionary = dict(countriesNcaptitals)

In [88]: cncDictionary, type(cncDictionary)
Out[88]: ({'Germany': 'Berlin',
           'India': 'New Delhi',
           'UK': 'London',
           'US': 'Washington'},
          dict)

In [89]: cmp(d1,d2)
Out[89]: 0

In [90]: d1['e'] = 'elephant'
```

In [91]:
```
cmp(d1,d2)
```
Out[91]:
```
1
```

In [92]:
```
d2['e'] = 'eagle'
```

In [93]:
```
len(d1), len(d2)
```
Out[93]:
```
(4, 4)
```

In [94]:
```
cmp(d1,d2)
```
Out[94]:
```
1
```

In [95]:
```
d4 ={}
d4.fromkeys(d1)  # to extract the keys of d1, and place them for d4
```
Out[95]:
```
{'a': None, 'b': None, 'c': None, 'e': None}
```

In [96]:
```
d5 ={}
d5.fromkeys(d1, 'Python') # To place a default value, instead of None
```
Out[96]:
```
{'a': 'Python', 'b': 'Python', 'c': 'Python', 'e': 'Python'}
```

dictionary Values can't be extracted in the same way

In [97]:
```
print dir(d1)
```
```
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc
__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__
gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__
ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy',
 'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'iterkeys', 'itervalue
s', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems',
 'viewkeys', 'viewvalues']
```

In [98]:
```
d1
```
Out[98]:
```
{'a': 'apple', 'b': 'banana', 'c': 'canada', 'e': 'elephant'}
```

In [99]:
```
d1.get('a')
```
Out[99]:
```
'apple'
```

In [100]:
```
d1.has_key('a')
```
Out[100]:
```
True
```

In [101]:
```
d1.has_key('c')
```
Out[101]:
```
True
```

Assignment 1: Explore the usage of these dictionaries attributes: 'pop', 'popitem'

Assignment 2: Write a script to take the names of our five friends in a list, and their designation s in a separate list. ... Then, create a dictionary, containing their name, designation pairs

Assignment 3: Write a script to get the letter frequency from a given sentence. The first three letters are repeated.

```
In [102]: d1
Out[102]: {'a': 'apple', 'b': 'banana', 'c': 'canada', 'e': 'elephant'}

In [103]: d1.setdefault('a', None)  # works same as indexing a key, when key is present
Out[103]: 'apple'

In [104]: d1.setdefault('f', None) # return None, if the key is not present

In [105]: d1.setdefault('d', 'donut') # to set a specific value to that key, if key is n
          ot present
Out[105]: 'donut'

In [106]: d1
Out[106]: {'a': 'apple',
           'b': 'banana',
           'c': 'canada',
           'd': 'donut',
           'e': 'elephant',
           'f': None}

In [107]: d1.values()     # results a list
Out[107]: ['apple', 'canada', 'banana', 'elephant', 'donut', None]

In [108]: d1.viewvalues()   # results a dict_item
Out[108]: dict_values(['apple', 'canada', 'banana', 'elephant', 'donut', None])

In [109]: d1.keys()
Out[109]: ['a', 'c', 'b', 'e', 'd', 'f']

In [110]: d1.viewkeys()
Out[110]: dict_keys(['a', 'c', 'b', 'e', 'd', 'f'])
```

```
In [111]: d1.items()

Out[111]: [('a', 'apple'),
           ('c', 'canada'),
           ('b', 'banana'),
           ('e', 'elephant'),
           ('d', 'donut'),
           ('f', None)]
```

```
In [112]: d1.viewitems()

Out[112]: dict_items([('a', 'apple'), ('c', 'canada'), ('b', 'banana'), ('e', 'elephan
          t'), ('d', 'donut'), ('f', None)])
```

```
In [113]: items = d1.iteritems()  # returns items as a generator; need to iterate with n
          ext() to get the items
```

```
In [114]: items

Out[114]: <dictionary-itemiterator at 0x4563360>
```

```
In [115]: items.next()

Out[115]: ('a', 'apple')
```

d1.iterkeys() and d1.itervalues() will work in the same way.

```
In [116]: d1= {True: 'one', 2: '2222', 3: 'three', 'b': 'ball'}
```

```
In [117]: d1.pop('abcd', None) # returns None, if the key is not present
```

```
In [118]: d1.pop('abcd', "No Such Key") # To return default statement, in the absence of
           key

Out[118]: 'No Such Key'
```

```
In [119]: d1.pop('abcd')

          ---------------------------------------------------------------------------
          KeyError                                  Traceback (most recent call last)
          <ipython-input-119-2c1b5f98b6d5> in <module>()
          ----> 1 d1.pop('abcd')

          KeyError: 'abcd'
```

```
In [120]: d1 = {True: 'one', 2: '2222', 3: 'three', 'b': 'ball'}
```

```
In [121]: key,value = d1.popitem()
In [122]: print key, value
```

```
True one
```

```
In [123]: key,value = d1.popitem(); print key,value
```

```
2 2222
```

```
In [124]: key,value = d1.popitem(); print key,value
```

```
3 three
```

```
In [125]: key,value = d1.popitem(); print key,value  # No more items left
```

```
b ball
```

In [126]:
```
# characterFrequencyAnalysis.py
"""
        Purpose: To count the number of times, each character occurred in the
 sentence.
        Output: Each character and its occurrence count, as a pair.
"""

#sentence = "It always seem impossible, until it is achieved!"
sentence = raw_input("Enter a Quote: ")
count = {} # empty dictionary
for character in sentence:
     count[character] = count.get(character, 0) + 1

print "character: occurrenceFrequency \n"

#print count

#for key,value in count.items():
#    print key, value

for item in count.items():
        print item
```

```
Enter a Quote: It always seem impossible, until it is achieved!
character: occurrenceFrequency

('!', 1)
(' ', 7)
(',', 1)
('I', 1)
('a', 3)
('c', 1)
('b', 1)
('e', 5)
('d', 1)
('i', 6)
('h', 1)
('m', 2)
('l', 3)
('o', 1)
('n', 1)
('p', 1)
('s', 5)
('u', 1)
('t', 3)
('w', 1)
('v', 1)
('y', 1)
```

Memoization

To store the values which are already compiled, in cache, to optimize the time consumption

```
In [128]: alreadyknown = {0: 0, 1: 1}
```

```
In [129]: def fib(n):
              if n not in alreadyknown:
                  new_value = fib(n-1) + fib(n-2)
                  alreadyknown[n] = new_value
              return alreadyknown[n]
```

```
In [130]: print fib(20)

          6765
```

Interview Question 2: What is memoization. How to achieve it in dictionaries?

Ordered Dictionary

```
In [131]: d = {'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2}  # regular unsorted dict
          ionary
```

```
In [132]: import collections
```

```
In [133]: collections.OrderedDict(sorted(d.items(), key=lambda t: t[0])) # Sorted by key

Out[133]: OrderedDict([('apple', 4), ('banana', 3), ('orange', 2), ('pear', 1)])
```

```
In [134]: collections.OrderedDict(sorted(d.items(), key=lambda t: t[1])) # Sorted by Val
          ue

Out[134]: OrderedDict([('pear', 1), ('orange', 2), ('banana', 3), ('apple', 4)])
```

```
In [135]: collections.OrderedDict(sorted(d.items(), key=lambda t: len(t[0]))) # Sorted b
          y length of key string.

Out[135]: OrderedDict([('pear', 1), ('apple', 4), ('orange', 2), ('banana', 3)])
```

Assignment 4: Use collections.OrdererDict to get an ordered dictionary. Try to do some example

Interview Question 3: How to sort a dictionary based on the length of the key?

```
In [136]: d

Out[136]: {'apple': 4, 'banana': 3, 'orange': 2, 'pear': 1}
```

```
In [139]: sorted(d.items(), key = lambda t: len(t[0]))

Out[139]: [('pear', 1), ('apple', 4), ('orange', 2), ('banana', 3)]
```