Dictionary and Sets

A dictionary is a sequence of items. Each item is a pair made of a key and a value. Dictionaries are not sorted. You can access to the list of keys or values independently.

```
>>> d = {'first':'string value', 'second':[1,2]}
>>> d.keys()
['first', 'second']
>>> d.values()
['string value', [1, 2]]
```
You can access to the value of a given key as follows:

```
>>> d['first']
'string value'
```

Warning
You cannot have duplicate keys in a dictionary

Warning
dictionary have no concept of order among elements

Methods to query information
In addition to keys and values methods, there is also the items method that returns a list of items of the form (key, value). The items are not returned in any particular order:

```
>>> d = {'first':'string value', 'second':[1,2]}
>>> d.items()
[('a', 'string value'), ('b', [1, 2])]
```

You can check for the existence of a specific key with has_key:

```
>>> d.has_key('first')
True
```

The expression d.has_key(k) is equivalent to k in d. The choice of which to use is largely a matter of taste.

In order to get the value corresponding to a specific key, use get or pop:

```
>>> d.get('first')  # this method can set an optional value, if the key is not found
'string value'
```

It is useful for things like adding up numbers:

```
sum[value] = sum.get(value, 0) + 1
```

The difference between get and pop is that pop also removes the corresponding item from the dictionary:

```
>>> d.pop('first')
'string value'
>>> d
{'second': [1, 2]}
```

Finally, popitem removes and returns a pair (key, value); you do not choose which one because a dictionary is not sorted

```
>>> d.popitem()
('a', 'string value')
>>> d
{'second': [1, 2]}
```

Methods to create new dictionary
Since dictionaries (like other sequences) are objects, you should be careful when using the affectation sign:

```
>>> d1 = {'a': [1,2]}
>>> d2 = d1
>>> d2['a'] = [1,2,3,4]
>>> d1['a]
[1,2,3,4]
```

To create a new object, use the copy method (shallow copy):
```
>>> d2 = d1.copy()
```

You can clear a dictionary (i.e., remove all its items) using the clear() method:
```
>>> d2.clear()
{}
```

The clear() method deletes all items whereas del() deletes just one:
```
>>> d = {'a':1, 'b':2, 'c':3}
>>> del d['a']
>>> d.clear()
```

Create a new item with default value (if not provided, None is the default):

```
>>> d2.setdefault('third', '')
>>> d2['third']
''
```

Create a dictionary given a set of keys:

```
>>> d2.fromkeys(['first', 'second'])
```

another syntax is to start from an empty dictionary:

```
>>> {}.fromkeys(['first', 'second'])
{'first': None, 'second': None}
```

Just keep in ,ind thqt the fromkeys() method creates a new dictionary with the given keys, each with a default corresponding value of None.

Combining dictionaries

Given 2 dictionaries d1 and d2, you can add all pairs of key/value from d2 into d1 by using the update method (instead of looping and assigning each pair yourself:

```
>>> d1 = {'a':1}
>>> d2 = {'a':2; 'b':2}
>>> d1.update(d2)
>>> d1['a']
2

>>> d2['b']
2
```

The items in the supplied dictionary are added to the old one, overwriting any items there with the same keys.

iterators

Dictionary provides iterators over values, keys or items:

```
>>> [x for x in t.itervalues()]
['string value', [1, 2]]
>>>
>>> [x for x in t.iterkeys()]
['first', 'csecond']
>>> [x for x in t.iteritems()]
[('a', 'string value'), ('b', [1, 2])]
```

```
stocks = {
        'IBM': 146.48,
        'MSFT':44.11,
        'CSCO':25.54
}

#print out all the keys
for c in stocks:
        print(c)

#print key n values
for k, v in stocks.items():
        print("Code : {0}, Value : {1}".format(k, v))

main()
```

```
MSFT
IBM
CSCO
Code : MSFT, Value : 44.11
Code : IBM, Value : 146.48
Code : CSCO, Value : 25.54
```

Sets are constructed from a sequence (or some other iterable object). Since sets cannot have duplicated, there are usually used to build sequence of unique items (e.g., set of identifiers).

4.5.1. Quick example

```
>>> a = set([1, 2, 3, 4])
>>> b = set([3, 4, 5, 6])
>>> a | b # Union
{1, 2, 3, 4, 5, 6}
>>> a & b # Intersection
{3, 4}
>>> a < b # Subset
False
>>> a - b # Difference
{1, 2}
>>> a ^ b # Symmetric Difference
{1, 2, 5, 6}
```

Note
the intersection, subset, difference and symmetric difference can be called with method rather that symbols. See below for examples.

Ordering
Just as with dictionaries, the ordering of set elements is quite arbitrary, and shouldn't be relied on.

Operators
As mentionned in the quick example section, each operator is associated to a symbol (e.g., &) and a method name (e.g. union).

```
>>> a = set([1, 2, 3])
>>> b = set([2, 3, 4])
>>> c = a.intersection(b) # equivalent to c = a & b
>>> a.intersection(b)
set([2, 3])
>>> c.issubset(a)
True
>>> c <= a
True
>>> c.issuperset(a)
False
>>> c >= a
False
>>> a.difference(b)
set([1])
>>> a - b
set([1])
>>> a.symmetric_difference(b)
set([1, 4])
>>> a ^ b
set([1, 4])
```

You can also copy a set using the copy method:

```
>>> a.copy()
set([1, 2, 3])
```