Content Delivered in class_7_05-August-2016

- Chapter 4: Collections
  - Sets
    - Mutability of sets
    - frozenset
    - Orderedset

Interview Questions Discussed:

Interview Question 1: What is the simplest way to remove duplicates in a list?

Interview Question 2: what is the difference between discard, pop and remove methods of set

Interview Question 3: what is the result of set1 = {1, 'Python', True}

Assignments Given:

Assignment 1 : Explore the differences between set.remove() vs set.discard() vs set.pop()

Assignment 2: Explore the differences between set.update() and set.add()

Assignment 3: Try to get a sorted set from the given set, using orderedset module

---

sets

- sets are unordered; Then can't be indexed
- sets doesn't store duplicates; It will discard the two and other consequent occurrences of the same element.
- denoted with {}

In [4]:
```
s1 = set()    # empty set
```

In [5]:
```
print s1, type(s1)
```

```
set([]) <type 'set'>
```

In [6]:
```
s1 = {1,2,2, 3,4,5,3,5}  # simple set
```

In [7]:
```
s1
```

Out[7]:
```
{1, 2, 3, 4, 5}
```

In [8]:
```
s2 = {1,2,[1,2], (1,2), 1,2}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-d0aa9a001444> in <module>()
----> 1 s2 = {1,2,[1,2], (1,2), 1,2}

TypeError: unhashable type: 'list'
```

elements in a set must be immutable only.

```
mutable  - list
Immutable - tuple, string
```

In [9]:
```
s2 = {1,2,(1,2), (1,2), 1,2}
```

In [10]:
```
s2
```

Out[10]: {1, 2, (1, 2)}

Interview Question 1: What is the simplest way to remove duplicates in a list?

Ans list(set(list1))

In [12]:
```
myList = [1,2,3,2,3,45]
```

In [13]:
```
myList = list(set(myList))   # set() and list() are buit-in functions
```

In [14]:
```
print myList
```

```
[1, 2, 3, 45]
```

In [15]:
```
s3  = {'Apple', 'Mango', 'Banana', 12, 'Bnana', 'Mango'}
```

In [16]:
```
s3      # observe the odred in which they are display
```

Out[16]: {12, 'Apple', 'Banana', 'Bnana', 'Mango'}

In [17]:
```
s3
```

Out[17]: {12, 'Apple', 'Banana', 'Bnana', 'Mango'}

In [18]:
```
print dir(s3)
```

```
['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__iand__', '__init__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le
__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduc
e__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__seta
ttr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'a
dd', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersec
tion', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop',
 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'u
pdate']
```

sets can't be indexed

Arithmetic Operations are not supported by sets

In [19]:
```
s4 = {'Mercedes', 'Toyota', 'Maruthi', 'Hyundai'}
```

In [20]:
```
s3+s4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-eb18ff61ffe4> in <module>()
----> 1 s3+s4

TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

In [21]:
```
s3-s4  # It means to get elements of s3, which are not present in s4
```

Out[21]:
```
{12, 'Apple', 'Banana', 'Bnana', 'Mango'}
```

In [22]:
```
s5 = {'suzuki', 'Renault', 'Toyota'}
```

In [23]:
```
s4-s5
```

Out[23]:
```
{'Hyundai', 'Maruthi', 'Mercedes'}
```

In [24]:
```
s5-s4
```

Out[24]:
```
{'Renault', 'suzuki'}
```

In [25]:
```
countries = set(['India', 'Afganistan', 'Sri Lanka', 'Nepal'])
```

In [26]:
```
type(countries)   # list to set
```

Out[26]:
```
set
```

In [27]:
```
brics = set(('Brazil', 'Russia', 'India', 'China', 'South Africa'))
```

In [28]:
```
type(brics)  # tuple to set
```

Out[28]:
```
set
```

In [29]:
```
ch = set('Python Programming')
```

In [30]:
```
type(ch)  # string to set of characters
```

Out[30]:
```
set
```

In [31]:
```
ch
```

Out[31]:
```
{' ', 'P', 'a', 'g', 'h', 'i', 'm', 'n', 'o', 'r', 't', 'y'}
```

In [32]:
```
import sets
```

```
c:\python27\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: th
e sets module is deprecated
  if __name__ == '__main__':
```

In [34]:
```
asean = sets.Set(['Myanmar', 'Indonesia', 'Malaysia', 'Philiphines'])
```

In [35]:
```
asean    # list to sets
```

Out[35]:
```
Set(['Malaysia', 'Philiphines', 'Indonesia', 'Myanmar'])
```

In [36]:
```
africa = sets.Set(['south Africa', 'Mozambique', ['Moracco', 'tunisia'], ('ken
ya', 'sudan')]) # list contains a list and tuple in it
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-1a082dc9a7db> in <module>()
----> 1 africa = sets.Set(['south Africa', 'Mozambique', ['Moracco', 'tunisi
a'], ('kenya', 'sudan')]) # list contains a list and tuple in it

c:\python27\lib\sets.py in __init__(self, iterable)
    412         self._data = {}
    413         if iterable is not None:
--> 414             self._update(iterable)
    415
    416     def __getstate__(self):

c:\python27\lib\sets.py in _update(self, iterable)
    357                 try:
    358                     for element in it:
--> 359                         data[element] = value
    360                     return
    361                 except TypeError:

TypeError: unhashable type: 'list'
```

In [38]:
```
africa = sets.Set(tuple(['south Africa', 'Mozambique', ['Moracco', 'tunisia'],
 ('kenya', 'sudan')]))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-987e43c2cb08> in <module>()
----> 1 africa = sets.Set(tuple(['south Africa', 'Mozambique', ['Moracco', 't
unisia'], ('kenya', 'sudan')]))

c:\python27\lib\sets.py in __init__(self, iterable)
    412         self._data = {}
    413         if iterable is not None:
--> 414             self._update(iterable)
    415
    416     def __getstate__(self):

c:\python27\lib\sets.py in _update(self, iterable)
    357                 try:
    358                     for element in it:
--> 359                         data[element] = value
    360                     return
    361                 except TypeError:

TypeError: unhashable type: 'list'
```

In [39]:
```
engineers = set(['John', 'Jane', 'Jack', 'Janice'])
```

In [41]:
```
programmers = sets.Set({'Jack', 'Sam', 'Susan', 'Janice'})
```

In [42]:
```
managers  = {'Jane', 'Jack', 'Susan', 'Zack'}
```

In [43]:
```
type(engineers), type(programmers), type(managers)
```

Out[43]: (set, sets.Set, set)

In [45]:
```
programmers = set(programmers)
```

In [46]:
```
type(engineers), type(programmers), type(managers)
```

Out[46]: (set, set, set)

```
|  - union operator
&  - Intersection operator
-  - difference operator
```

In [47]:
```
employees = engineers | programmers | managers
```

In [48]:
```
employees
```

Out[48]: {'Jack', 'Jane', 'Janice', 'John', 'Sam', 'Susan', 'Zack'}

In [49]:
```
engg_managers = engineers & managers
```

In [50]:
```
engg_managers
```

Out[50]:
```
{'Jack', 'Jane'}
```

In [51]:
```
onlyManagers = managers - engineers - programmers # same as (managers - engine
ers) - programmers
```

In [52]:
```
onlyManagers
```

Out[52]:
```
{'Zack'}
```

In [53]:
```
onlyEngineers = engineers - managers - programmers
```

In [54]:
```
onlyEngineers
```

Out[54]:
```
{'John'}
```

Mutability of sets

In [55]:
```
engineers.add('Yash')  # add - to add an element to the set
```

In [56]:
```
engineers
```

Out[56]:
```
{'Jack', 'Jane', 'Janice', 'John', 'Yash'}
```

In [57]:
```
employees.issuperset(engineers)
```

Out[57]:
```
False
```

In [58]:
```
employees
```

Out[58]:
```
{'Jack', 'Jane', 'Janice', 'John', 'Sam', 'Susan', 'Zack'}
```

In [59]:
```
employees.discard('Susan')
```

In [60]:
```
employees
```

Out[60]:
```
{'Jack', 'Jane', 'Janice', 'John', 'Sam', 'Zack'}
```

In [61]:
```
employees.discard('Yale')  # didn't result any exxecption, even though 'Yale'
 is not present in set
```

frozenset

set is a mutable object; elements in a set can be modified
frozenset is an immutable object; elements in a frozenset can't be modified

In [62]:
```
vetoCountries = set(['US', 'UK', 'Russia', 'China', 'France'])
```

In [63]:
```
type(vetoCountries)
```

Out[63]:
```
set
```

In [65]:
```
vetoCountries = frozenset(['US', 'UK', 'Russia', 'China', 'France'])
```

In [66]:
```
type(vetoCountries)
```

Out[66]:
```
frozenset
```

In [67]:
```
print dir(vetoCountries)
```

```
['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__',
 '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__',
 '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
 '__subclasshook__', '__xor__', 'copy', 'difference', 'intersection', 'isdisj
oint', 'issubset', 'issuperset', 'symmetric_difference', 'union']
```

In [69]:
```
fruits = {'Mango', 'Apple', 'Papaya'}
```

In [70]:
```
vegetables = {'Beetroot', 'cabbage', 'Carrot', 'Carrot'}
```

In [71]:
```
fruits.union(vegetables)
```

Out[71]:
```
{'Apple', 'Beetroot', 'Carrot', 'Mango', 'Papaya', 'cabbage'}
```

In [72]:
```
fruitsAndVegetables = fruits.union(vegetables)
```

In [73]:
```
fruits.update('tomato')
```

In [74]:
```
fruits
```

Out[74]:
```
{'Apple', 'Mango', 'Papaya', 'a', 'm', 'o', 't'}
```

In [75]:
```
fruits.update(['tomato'])
```

In [76]:
```
fruits
```

Out[76]:
```
{'Apple', 'Mango', 'Papaya', 'a', 'm', 'o', 't', 'tomato'}
```

In [77]:
```
fruits.discard('a')
```

In [78]:
```
fruits
```

Out[78]:
```
{'Apple', 'Mango', 'Papaya', 'm', 'o', 't', 'tomato'}
```

In [79]:
```
fruits.discard('m','0', 't')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-79-9b36542a6e75> in <module>()
----> 1 fruits.discard('m','0', 't')

TypeError: discard() takes exactly one argument (3 given)
```

In [81]:
```
fruits.discard(('m','0', 't'))  # It checks for this tuple element in set
```

In [82]:
```
fruits
```

Out[82]:
```
{'Apple', 'Mango', 'Papaya', 'm', 'o', 't', 'tomato'}
```

In [83]:
```
vegetables.update(['tomato', 'watermelon'])
```

In [84]:
```
vegetables
```

Out[84]:
```
{'Beetroot', 'Carrot', 'cabbage', 'tomato', 'watermelon'}
```

In [85]:
```
fruits.intersection(vegetables)
```

Out[85]:
```
{'tomato'}
```

In [86]:
```
vegetables.intersection(fruits)
```

Out[86]:
```
{'tomato'}
```

In [87]:
```
fruits - vegetables
```

Out[87]:
```
{'Apple', 'Mango', 'Papaya', 'm', 'o', 't'}
```

In [88]:
```
vegetables - fruits
```

Out[88]:
```
{'Beetroot', 'Carrot', 'cabbage', 'watermelon'}
```

In [89]:
```
fruits.intersection(vegetables) == vegetables.intersection(fruits)
```

Out[89]:
```
True
```

In [90]:
```
fruits - vegetables  != vegetables - fruits
```

Out[90]:
```
True
```

set difference is noyt cumulative; whereas intersection attribute of set is cumulative. Intersection results in the common elements among the sets given

```
In [91]: fruits.isdisjoint(vegetables)  # no, there is a common element
Out[91]: False

In [92]: fruits.isdisjoint(vetoCountries)  #yes, there is no common element
Out[92]: True

In [93]: fruits.pop()
Out[93]: 'tomato'

In [94]: fruits
Out[94]: {'Apple', 'Mango', 'Papaya', 'm', 'o', 't'}

In [97]: fruits.pop()
Out[97]: 'Papaya'

In [98]: fruits.remove('t')

In [99]: fruits
Out[99]: {'Apple', 'Mango', 'm', 'o'}
```

Interview Question 2: what is the difference between discard, pop and remove methods of set

remove() - to delete a specific element (not with position, but the element itself).

Assignment 1 : Explore the differences between set.remove() vs set.discard() vs set.pop()

Assignment 2: Explore the differences between set.update() and set.add()

```
In [101]: asean
Out[101]: Set(['Malaysia', 'Philiphines', 'Indonesia', 'Myanmar'])

In [102]: asean.pop()  # deletes an elemnt, in random; So, not preferred
Out[102]: 'Malaysia'

In [103]: asean.remove('Myanmar') # delete the given element
```

remove() can't delete multiple elements

set.remove() vs set.discard() vs set.pop()

```
set.remove()   - Used to remove an element. Thorws KeyError, if the specifed elemen
t is not present
set.discard()  - Used to remove an element. Doesn't raise any error, if specified e
lement is not present.
set.pop()      - Used to remove and RETURN a random element from the set. Raises ke
yError, if no element is present.
```

for sets A and B, symmetric diiference is (A-B) | (B-A)

In [104]:
```
fruits = {'Mango', 'Apple', 'Papaya', 'tomato'}
```

In [105]:
```
vegetables = {'Beetroot', 'cabbage', 'tomato', 'Carrot', 'Carrot'}
```

In [106]:
```
fruits.symmetric_difference(vegetables)
```

Out[106]:
```
{'Apple', 'Beetroot', 'Carrot', 'Mango', 'Papaya', 'cabbage'}
```

In [107]:
```
(fruits - vegetables) | (vegetables - fruits)
```

Out[107]:
```
{'Apple', 'Beetroot', 'Carrot', 'Mango', 'Papaya', 'cabbage'}
```

symmetric_difference() is cumulative

Interview Question 3: what is the result of set1 = {1, 'Python', True}

In [108]:
```
set1 = {1, 'Python', True}
```

In [109]:
```
set1
```

Out[109]:
```
{True, 'Python'}
```

'True' is preferred as it is built-in object

In [110]:
```
all(set1)
```

Out[110]:
```
True
```

In [112]:
```
set2 = {10, 10.9, 0.01, 0, 'Prog', None}
```

In [113]:
```
all(set2)
```

Out[113]: False

In [114]:
```
any(set2)
```

Out[114]: True

In [115]:
```
for i in set2:
    print i
```

```
0
10
None
10.9
Prog
0.01
```

In [117]:
```
[i for i in set2]
```

Out[117]: [0, 10, None, 10.9, 'Prog', 0.01]

In [118]:
```
for i,j in enumerate(set2):   # enumerate stores teh index of the elements iterated.
    print i,j
```

```
0 0
1 10
2 None
3 10.9
4 Prog
5 0.01
```

In [122]:
```
max(set2), min(set2)
```

Out[122]: ('Prog', None)

In [123]:
```
a = None; print type(a)
```

```
<type 'NoneType'>
```

sorted() is not applicable to sets; whereas len() is applicable

In [124]:
```
len(set2)
```

Out[124]: 6

Orderedset

- Used to store elements in an ascending order
- This is a module, to be imported
- This module doesn;t come with standard library
- It must be installed using the command

```
pip install orderedset
```

In [126]:
```
import orderedset
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-126-6c001215ec47> in <module>()
----> 1 import orderedset

ImportError: No module named orderedset
```

In [127]:
```
import os; os.system('pip install orderedset')  # To install 'orderedset' modu
le within the interpreter
```

Out[127]: 1

We can discuss more about 'os' module in modules chapter

In [ ]:
```
oset = orderedset.OrderedSet([11, 2, 3])
```

In [ ]:
```
oset        # orderedset.OrderedSet() ensures that the assigned order of the s
et is retained.
```

In [ ]:
```
oset | [5,4,3,2,1]
```

In [ ]:
```
oset | {3, 33, 333}
```

Assignment 3: Try to get a sorted set from the given set, using orderedset module