Setp0 - Sets

May 8, 2018

0.0.1 Sets

A set contains an unordered collection of unique and immutable objects. The set data type is, as the name implies, a Python implementation of the sets as they are known from mathematics. This explains, why sets unlike lists or tuples can't have multiple occurrences of the same element

0.0.2 Creating sets

```
In [7]: # Creating an empty set
        languages = set()
        print type(languages), languages

        languages = {'Python', 'R', 'SAS', 'Julia'}
        print type(languages), languages

        # set of mixed datatypes
        mixed_set = {"Python", (2.7, 3.4)}
        print type(mixed_set), languages
```

```
<type 'set'> set([])
<type 'set'> set(['SAS', 'Python', 'R', 'Julia'])
<type 'set'> set(['SAS', 'Python', 'R', 'Julia'])
```

0.0.3 Accessing set elements

```
In [4]: print list(languages)[0]
        print list(languages)[0:3]
```

```
SAS
['SAS', 'Python', 'R']
```

0.0.4 Changing a set in Python

Although sets are mutable, indexing on them will have no meaning due to the fact that they are unordered. So sets do not support accessing or changing an item/element using indexing or slicing. The add() method can be used to add single element and update() method for adding multiple elements. Note that the update() method can take the argument in the format of tuples, lists, strings or other sets. However, in all cases the duplicates are ignored.

```
In [5]: # initialize a set
        languages = {'Python', 'R'}
        print(languages)

        # add an element
        languages.add('SAS')
        print(languages)

        # add multiple elements
        languages.update(['Julia','SPSS'])
        print(languages)

        # add list and set
        languages.update(['Java','C'], {'Machine Learning','Data Science','AI'})
        print(languages)
```

```
set(['Python', 'R'])
set(['Python', 'SAS', 'R'])
set(['Python', 'SAS', 'R', 'Julia', 'SPSS'])
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'AI', 'R', 'SAS', 'Machine Learnin
```

0.0.5 Removing items from set

The discard() or remove() method can be used to remove a particular item from set. The fundamental difference between discard() and remove() is that the first do not take any action if the item does not exist in the set, whereas remove() will raises an error in such scenario.

```
In [6]: # remove an element
        languages.remove('AI')
        print(languages)

        # discard an element, although AI has already been removed discard will not throw an e
        languages.discard('AI')
        print(languages)

        # Pop will remove a random item from set
        print "Removed:", (languages.pop()), "from", languages
```

```
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS', 'Machine Learning'])
set(['C', 'Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS', 'Machine Learning'])
Removed: C from set(['Java', 'Python', 'Data Science', 'Julia', 'SPSS', 'R', 'SAS', 'Machine Le
```

0.1 Set operations

As discussed earlier, sets allow us to use mathematical set operations such as union, intersection, difference and symmetric difference. We can achieve this with the help of operators or methods.

0.1.1 Set Union

Union of two sets A and B will result to a set of all items combined from both sets. There are two ways of to perform union operation 1) Using | operator, 2) using union() method

```
In [1]: # initialize A and B
        A = {1, 2, 3, 4, 5}
        B = {4, 5, 6, 7, 8}

        # use | operator
        print "Union of A | B", A|B

        # alternative we can use union()
        A.union(B)
```

```
Union of A | B set([1, 2, 3, 4, 5, 6, 7, 8])
```

```
Out[1]: {1, 2, 3, 4, 5, 6, 7, 8}
```

0.1.2 Set Intersection

Intersection of two sets A and B will result to a set of items that exists or common in both sets. There are two ways to achieve intersection operation 1) using & operator, 2) using intersection() method

```
In [2]: # use & operator
        print "Intersection of A & B", A & B

        # alternative we can use intersection()
        print A.intersection(B)
```

```
Intersection of A & B set([4, 5])
set([4, 5])
```

0.1.3 Set Difference

Difference of two sets A and B (i.e., A - B) will result into a set of items which exists only in A and not in B. There are two ways to perform difference operation 1) using '–' operator, and 2) using difference() method

```
In [3]: # use - operator on A
        print "Difference of A - B", A - B

        # alternative we can use difference()
        print A.difference(B)
```

```
Difference of A - B set([1, 2, 3])
set([1, 2, 3])
```

0.1.4 Set Symmetric Difference

Symmetric difference of two sets A and B is a set of items from both sets that are not common. There are two ways to perform symmetric difference 1) using ^ operator, and 2) using symmetric_difference() mehtod

```
In [4]: # use ^ operator
        print "Symmetric difference of A ^ B", A ^ B

        # alternative we can use symmetric_difference()
        A.symmetric_difference(B)

Symmetric difference of A ^ B set([1, 2, 3, 6, 7, 8])


Out[4]: {1, 2, 3, 6, 7, 8}
```

0.1.5 Basic operations

```
In [9]: # Return a shallow copy of a set
        lang = languages.copy()
        print languages
        print lang

        # initialize A and B
        A = {1, 2, 3, 4, 5}
        B = {4, 5, 6, 7, 8}

        print A.isdisjoint(B)   # True, when two sets have a null intersection
        print A.issubset(B)     # True, when another set contains this set
        print A.issuperset(B)   # True, when this set contains another set
        sorted(B)               # Return a new sorted list
        print sum(A)           # Retrun the sum of all items
        print len(A)           # Return the length
        print min(A)           # Return the largest item
        print max(A)           # Return the smallest item

set(['SAS', 'Python', 'R', 'Julia'])
set(['SAS', 'Python', 'R', 'Julia'])
False
False
False
15
5
1
```

Reference: Mastering machine learning with python in six-steps