Python Set Operation

Anoop S Babu
Faculty Associate
Dept. of Computer Science & Engineering
bsanoop@am.amrita.edu

Sets – Membership operator

- Checks the existence of an element

```
>>> numbers = {2,4,6,8}

>>> 2 in numbers            # check if 2 is in numbers
True

>>> 5 not in numbers        # check if 5 is not in numbers
True
```

Set operations

- union()
- Intersection()
- Difference()
- Symmetric_difference()
- Isdisjoint()
- Issubset()
- Issuperset()
- Intersection_update()
- Difference_update()
- Symmetric_difference_update()

Union

- Using union() method

```
>>> s1 = {1,3,5}
>>> s2 = {2,4,5,6}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6}
```

- Using union operator - |

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6}
```

- Returns a new set with elements from all sets

Intersection

- Using intersection() method

```
>>> s1 = {1,3,5}
>>> s2 = {6,3,7,5,8}
>>> s1.intersection(s2)
{3, 5}
```

- Using intersection operator - &

```
>>> s1 & s2
{3, 5}
```

- Returns a new set with elements common from all sets

Difference

- Using difference() method

```
>>> s1 = {1,3,5}
>>> s2 = {6,3,5,7,8}
>>> s1.difference(s2)
{1}
>>> s2.difference(s1)
{8, 6, 7}
```

- Using difference operator – '-'

```
>>> s1 - s2
{1}
>>> s2 - s1
{8, 6, 7}
```

- Returns a new set with elements not in other

symmetric_difference

- Returns a new set with elements in either the set or other but not both.
- Using symmetric_difference() method

```
>>> s1 = {1,3,5}
>>> s2 = {6,3,5,7,8}
>>> s1.symmetric_difference(s2)
{1, 6, 7, 8}
```

- Using operator ^

```
>>> s1 ^ s2
{1, 6, 7, 8}
```

isdisjoint

- Returns true if two sets have no common elements
- Else returns false

```
>>> s1 = {1,3,5}
>>> s2 = {6,3,5,7,8}
>>> s3 = {2,4,6}
>>> s1.isdisjoint(s2)
False
>>> s1.isdisjoint(s3)
True
```

issubset

- Returns true if all elements in the set is in the other
- Using issubset() method

```
>>> s1 = {1,2}
>>> s2 = {2,4,1}
>>> s3 = {2,1}
>>> s4 = {3,5,6}
>>> s1.issubset(s2)
True
>>> s2.issubset(s1)
False
>>> s1.issubset(s3)
True
```

- Using operator <=

```
>>> s2 <=s1
False
>>> s2 <= s4
False
>>> s1 <=s2
True
>>> s1 <= s3
True
```

issuperset

- Returns true if all elements in the set is in the other
- Using issuperset() method

```
>>> s1 = {1,2}
>>> s2 = {2,4,1}
>>> s3 = {3,4,5}
>>> s2.issuperset(s1)
True
>>> s3.issuperset(s2)
False
```

- Using operator >=

```
>>> s2 >= s1
True
>>> s3 >= s2
False
```

Intersection_update

- Update the set with the elements in common

```
>>> s1 = {1,3,5}
>>> s2 = {6,3,5,7,8}
>>> s1.intersection_update(s2)
>>> print(s1)
{3, 5}
>>> print(s2)
{3, 5, 6, 7, 8}
```

Difference_update

- Update the set, removing elements found in others.

```
>>> s1 = {1,3,5}
>>> s2 = {6,3,4}
>>> s3 = {1,2}
>>> s1.difference_update(s2,s3)
>>> s1
{5}
```

symmetric_difference_update

- Update the set, keeping only elements found in either set, but not in both.

```
>>> s1 = {1,3,5}
>>> s2 = {6,3,5,7,8}
>>> s1.symmetric_difference_update(s2)
>>> print(s1)
{1, 6, 7, 8}
```

Frozen Sets

- They are Immutable unordered sets
- Created using frozenset() constructor
- Syntax is frozenset([iterable])

```
>>> fs = frozenset([2,3,2,4,1,3])
>>> fs
frozenset({1, 2, 3, 4})
```

- The following set operations are allowable:

  Union, intersection, difference, symmetric_difference, isdisjoint, issubset, issuperset and copy