016 Python Sets

Sets

A set object is an unordered collection of distinct hashable objects. It is commonly used in membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference.

Sets support x in the set, len(set), and for x in set like other collections. Set is an unordered collection and does not record element position or order of insertion. Sets do not support indexing, slicing, or other sequence-like behavior.

There are currently two built-in set types, set, and frozenset. The set type is mutable - the contents can be changed using methods like add() and remove(). Since it is mutable, it has no hash value and cannot be used as either a dictionary key or as an element of another set. The frozenset type is immutable and hashable - its contents cannot be altered after it is created; it can, therefore, be used as a dictionary key or as an element of another set.

Set: Commands

```
<set> = set()

<set>.add(<el>)
# Or: <set> |= {<el>}
<set>.update(<collection>)
# Or: <set> |= <set>
<set>  = <set>.union(<coll.>)
# Or: <set> | <set>
<set>  = <set>.intersection(<coll.>)
# Or: <set> & <set>
<set>  = <set>.difference(<coll.>)
# Or: <set> - <set>
<set>  = <set>.symmetric_difference(<coll.>)
# Or: <set> ^ <set>
<bool> = <set>.issubset(<coll.>)
# Or: <set> <= <set>
<bool> = <set>.issuperset(<coll.>)
# Or: <set> >= <set>

<el> = <set>.pop()
```

```
# Raises KeyError if empty.
<set>.remove(<el>)
# Raises KeyError if missing.
<set>.discard(<el>)
# Doesn't raise an error.
```

Frozen Set

- Is immutable and hashable.
- That means it can be used as a key in a dictionary or as an element in a set.

```
<frozenset> = frozenset(<collection>)
```

Create a set in Python:

```
>>> #A new empty set

>>> setx = set()

>>> print(setx)

set()

>>> #A non empty set

>>> n = set([0, 1, 2, 3, 4, 5])

>>> print(n)

{0, 1, 2, 3, 4, 5}

>>>
```

Iteration Over Sets:

We can move over each of the items in a set using a loop. However, since sets are unordered, it is undefined which order the iteration will follow.

```
>>> num_set = set([0, 1, 2, 3, 4, 5])

>>> for n in num_set:
  print(n)


  0

  1

  2

  3

  4

  5

>>>
```

Add member(s) in Python set:

```
>>> #A new empty set

>>> color_set = set()

>>> #Add a single member

>>> color_set.add("Red")

>>> print(color_set)

{'Red'}

>>> #Add multiple items

>>> color_set.update(["Blue", "Green"])

>>> print(color_set)

{'Red', 'Blue', 'Green'}

>>>
```

Remove item(s) from Python set:

pop(), remove() and discard() functions are used to remove individual item from a Python set. See the following examples :

pop() function:

```
>>> num_set = set([0, 1, 2, 3, 4, 5])

>>> num_set.pop()

0

>>> print(num_set)

{1, 2, 3, 4, 5}

>>> num_set.pop()

1

>>> print(num_set)

{2, 3, 4, 5}

>>>
```

remove() function:

```
>>> num_set = set([0, 1, 2, 3, 4, 5])

>>> num_set.remove(0)

>>> print(num_set)

{1, 2, 3, 4, 5}

>>>
```

discard() function:

```
>>> num_set = set([0, 1, 2, 3, 4, 5])

>>> num_set.discard(3)
>>> print(num_set)

{0, 1, 2, 4, 5}

>>>
```

Intersection of sets:

In mathematics, the intersection A ∩ B of two sets A and B is the set that contains all elements of A that also belong to B (or equivalently, all elements of B that also belong to A), but no other elements.

```
>>> #Intersection

>>> setx = set(["green", "blue"])

>>> sety = set(["blue", "yellow"])

>>> setz = setx & sety

>>> print(setz)

{'blue'}

>>>
```

Union of sets:

In set theory, the union (denoted by ∪) of a collection of sets is the set of all distinct elements in the collection. It is one of the fundamental operations through which sets can be combined and related to each other.

```
>>> #Union

>>> setx = set(["green", "blue"])

>>> sety = set(["blue", "yellow"])
>>> seta = setx | sety
>>> print (seta)
{'yellow', 'blue', 'green'}
>>>
```

Set difference:

```
>>> setx = set(["green", "blue"])
>>> sety = set(["blue", "yellow"])
>>> setz = setx & sety
>>> print(setz)
{'blue'}
>>> #Set difference
>>> setb = setx - setz
>>> print(setb)
{'green'}
>>>
```

Symmetric difference:

```
>>> setx = set(["green", "blue"])
>>> sety = set(["blue", "yellow"])
>>> #Symmetric difference
>>> setc = setx ^ sety
>>> print(setc)

{'yellow', 'green'}

>>>
```

issubset and issuperset:

```
>>> setx = set(["green", "blue"])

>>> sety = set(["blue", "yellow"])

>>> issubset = setx <= sety

>>> print(issubset)

False

>>> issuperset = setx >= sety

>>> print(issuperset)

False

>>>
```

More Example:

```
>>> setx = set(["green", "blue"])

>>> sety = set(["blue", "green"])

>>> issubset = setx <= sety

>>> print(issubset)

True

>>> issuperset = setx >= sety

>>> print(issuperset)

True
```

```
>>>
```

Shallow   of sets:

```
>>> setx = set(["green", "blue"])
>>> sety = set(["blue", "green"])
>>> #A shallow
>>> setd = setx. ()
>>> print(setd)
{'blue', 'green'}
>>>
```

Clear sets:

```
>>> setx = set(["green", "blue"])
>>> #Clear AKA empty AKA erase
>>> sete = setx. ()
>>> sete.clear()
>>> print(sete)
set()
>>>
```