8). issuperset()

```
In [2]: #This function returns True provided setobj1 contains all the elements of the stobj
        #syntax: setobj1.issuperset(setobj2)
```

```
In [4]: s1={10,20,30}
        s2={10,20}
        s3={10,15,25,35}
        s1.issuperset(s2)

Out[4]: True
```

```
In [6]: s2.issuperset(s1)

Out[6]: False
```

```
In [8]: s2.issuperset(s3)

Out[8]: False
```

```
In [10]: s3.issuperset(s1)

Out[10]: False
```

```
In [12]: s1.issuperset(s1) #Every set is super set to itself

Out[12]: True
```

```
In [14]: set().issuperset(set()) #***Every set is super set to itself

Out[14]: True
```

```
In [16]: set().issuperset(s1)

Out[16]: False
```

```
In [18]: s1.issuperset(set()) #s1 is Non-empty set greaterthan emptyset()

Out[18]: True
```

9).issubset()

```
In [21]: #syntax: setobj1.issubset(setobj2)
         #This function returns True provided all the elements of setobj1 present in setobj2
```

```
In [25]: s1={10,20,30}
         s2={10,20}
         s3={10,15,25,35}
         s2.issubset(s1)
```

```
Out[25]: True
```

```
In [27]: s1.issubset(s2)
```

```
Out[27]: False
```

```
In [29]: s1.issubset(s3)
```

```
Out[29]: False
```

```
In [31]: s3.issubset(s2)
```

```
Out[31]: False
```

```
In [35]: set().issubset({10,20,30}) #Emptyset subset of Non-emptyset
```

```
Out[35]: True
```

10). union()

```
In [38]: #This function is used for combining or mergining all the unique elements of setobj
         #and placed the elements-setobj3
```

```
In [40]: s1={10,20,30}
         s2={10,15,35}
         print(s1,type(s1))
         print(s2,type(s2))
```

```
{10, 20, 30} <class 'set'>
{10, 35, 15} <class 'set'>
```

```
In [42]: s3=s1.union(s2)
         print(s3,type(s3))
```

```
{35, 20, 10, 30, 15} <class 'set'>
```

```
In [46]: set().union(set()) #emptyset equal to emptyset
```

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

```
In [48]: set().union({10,20,30})
```

```
Out[48]: {10, 20, 30}
```

11). intersection()

```
In [51]: #syntax: setobj3=setobj1.intersection(setobj2)
         #This fuction used for obtaining the common elements between setobj1 and setobj2
         #If no common elecent found in between setobj1 and setobj2 then this fuction gives
```

```
In [53]: s1={10,20,30}
         s2={10,15,35}
```

```
print(s1,type(s1))
print(s2,type(s2))
```

```
{10, 20, 30} <class 'set'>
{10, 35, 15} <class 'set'>
```

In [59]:

```
s3=s1.intersection(s2)
print(s3,type(s3))
```

```
{10} <class 'set'>
```

In [61]:

```
set().intersection(set())
```

Out[61]:

```
set()
```

12). difference()

In [64]:

```
#syntax: setobj3=setobj1.difference(setobj2)
#This function removes common elecments from setobj1 and setobj2 takes the remainin
#and place to them in setobj3
```

In [66]:

```
s1={10,20,30}
s2={10,15,35}
print(s1,type(s1))
print(s2,type(s2))
```

```
{10, 20, 30} <class 'set'>
{10, 35, 15} <class 'set'>
```

In [68]:

```
s3=s1.difference(s2)
print(s3,type(s3))
```

```
{20, 30} <class 'set'>
```

In [70]:

```
s4=s2.difference(s1)
print(s4,type(s4))
```

```
{35, 15} <class 'set'>
```

In [72]:

```
set().difference({10,20,30})
```

Out[72]:

```
set()
```

In [76]:

```
s5={10,20,30}.difference(set())
print(s5,type(s5))
```

```
{10, 20, 30} <class 'set'>
```

In [78]:

```
s6={10,20,30}.difference({10,20,30})
print(s6,type(s6))
```

```
set() <class 'set'>
```

13). symmetric_difference() ** **Most Important** *

In [82]:

```
#syntax: setobj3=setobj1.symmetric_difference(setobj2)
#This function remove the common elecments from setobj1 and setobj2 and take the re
```

```
#elements from setobj1 and setobj2 and place them in setobj3
```

In [84]:
```
s1={10,20,30}
s2={10,15,35}
print(s1,type(s1))
print(s2,type(s2))
```

```
{10, 20, 30} <class 'set'>
{10, 35, 15} <class 'set'>
```

In [86]:
```
s3=s1.symmetric_difference(s2)
print(s3,type(s3))
```

```
{35, 15, 20, 30} <class 'set'>
```

In [88]:
```
s4=s2.symmetric_difference(s1)
print(s4,type(s4))
```

```
{35, 15, 20, 30} <class 'set'>
```

In [92]:
```
{11,12,13}.symmetric_difference({110,20,30})
```

Out[92]:
```
{11, 12, 13, 20, 30, 110}
```

In [94]:
```
s5=s1.union(s2).difference(s1.intersection(s2))
print(s4,type(s4))
```

```
{35, 15, 20, 30} <class 'set'>
```

14). update()

In [ ]:
```
#syntax: setobj3=setobj1.update(setobj2)
#This function is used adding all elements of setobj2 to setobj1 (setobj1 updated s
#and setobj3 contain nothing which is denoted as None
```

In [117…
```
s1={10,20,30}
s2={10,15,35}
s3=s1.update(s2)
print(s3,type(s3),id(s3))
```

```
None <class 'NoneType'> 140731861710800
```

In [119…
```
s1={10,20,30}
s2={10,20,25}
s1.update(s2)
print(s1)
```

```
{20, 25, 10, 30}
```

In [121…
```
s3=s1.difference_update(s2)
print(s1)
```

```
{30}
```

In [123…
```
print(s3)
```

```
None
```

In [125…

```
s3=s2.difference_update(s1)
print(s2)
```

```
{25, 10, 20}
```

In [127…

```
print(s3)
```

```
None
```

15). symmetric_difference_update()

In [132…

```
#syntax: setobj3=setobj1.symmetric_difference_update(setobj2)
#This function remove the common elecments from setobj1 and setobj2 and take the re
#elements from setobj1 and setobj2 and place them in setobj1 itself
```

In [134…

```
s1={10,20,30}
s2={10,15,35}
s3=s1.symmetric_difference_update(s2)
print(s1)
```

```
{35, 15, 20, 30}
```

In [136…

```
print(s3)
```

```
None
```

In [144…

```
s3=s2.symmetric_difference_update(s1)
print(s2)
```

```
{20, 10, 30}
```

In [140…

```
print(s3)
```

```
None
```

In [146…

```
s1={10,20,30}
s2={10,20,30}
s3=s1.symmetric_difference_update(s2)
print(s1)
```

```
set()
```

In [148…

```
help(set)
```

```
Help on class set in module builtins:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |  
 |  __or__(self, value, /)
 |      Return self|value.
 |  
 |  __rand__(self, value, /)
 |      Return value&self.
 |  
 |  __reduce__(...)
 |      Return state information for pickling.
 |  
 |  __repr__(self, /)
 |      Return repr(self).
 |  
 |  __ror__(self, value, /)
 |      Return value|self.
 |  
 |  __rsub__(self, value, /)
 |      Return value-self.
 |  
 |  __rxor__(self, value, /)
 |      Return value^self.
 |  
 |  __sizeof__(...)
 |      S.__sizeof__() -> size of S in memory, in bytes
 |  
 |  __sub__(self, value, /)
 |      Return self-value.
 |  
 |  __xor__(self, value, /)
 |      Return self^value.
 |  
 |  add(...)
 |      Add an element to a set.
 |  
 |      This has no effect if the element is already present.
 |  
 |  clear(...)
 |      Remove all elements from this set.
 |  
 |  copy(...)
 |      Return a shallow copy of a set.
 |  
 |  difference(...)
 |      Return the difference of two or more sets as a new set.
 |  
 |      (i.e. all elements that are in this set but not the others.)
 |  
 |  difference_update(...)
 |      Remove all elements of another set from this set.
 |  
 |  discard(...)
 |      Remove an element from a set if it is a member.
 |  
 |      Unlike set.remove(), the discard() method does not raise
 |      an exception when an element is missing from the set.
 |
 |  intersection(...)
 |      Return the intersection of two sets as a new set.
 |
 |      (i.e. all elements that are in both sets.)
 |
 |  intersection_update(...)
 |      Update a set with the intersection of itself and another.
 |
 |  isdisjoint(...)
 |      Return True if two sets have a null intersection.
 |
 |  issubset(self, other, /)
 |      Test whether every element in the set is in other.
 |
 |  issuperset(self, other, /)
 |      Test whether every element in other is in the set.
 |
 |  pop(...)
 |      Remove and return an arbitrary set element.
 |      Raises KeyError if the set is empty.
 |
 |  remove(...)
 |      Remove an element from a set; it must be a member.
 |
 |      If the element is not a member, raise a KeyError.
 |
 |  symmetric_difference(...)
 |      Return the symmetric difference of two sets as a new set.
 |
 |      (i.e. all elements that are in exactly one of the sets.)
 |
 |  symmetric_difference_update(...)
 |      Update a set with the symmetric difference of itself and another.
 |
 |  union(...)
 |      Return the union of sets as a new set.
 |
 |      (i.e. all elements that are in either set.)
 |
 |  update(...)
 |      Update a set with the union of itself and others.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...)
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

In [ ]: