**isdisjoint(*other*)**
Return True if the set has no elements in common with *other*. Sets are disjoint if and only if their intersection is the empty set.

```
>>> set1={1,2,3}
>>> set2={4,5,6}
>>> set1.isdisjoint(set2)
True
>>> set1.intersection(set2)
set()
>>> set3={1,24}
>>> set1.isdisjoint(set2)
True
>>> set1.isdisjoint(set3)
False
>>>
>>> java_students={'naresh','suresh'}
>>> python_students={'kishore','kiran'}
>>> if java_students.isdisjoint(python_students):
        print("no student of java is doing python")
else:
        print("some of the java students are doing python course")
no student of java is doing python
>>>
```

**issubset(*other*)**
set <= other
Test whether every element in the set is in *other*

```
>>> set1={1,2,3}
>>> set2={1,2,3,4,5}
>>> set1.issubset(set2)
True
>>> set3={1,4,5,6}
>>> set1.issubset(set3)
False
>>>
```

**issuperset(*other*)**
set >= other
Test whether every element in *other* is in the set.

```
>>> set1={1,2,3}
>>> set2={1,2,3,4,5}
>>> set2.issuperset(set1)
True
>>>
```

**Nested sets**
Sets of sets is called nested set or defining set inside set is called nested set

**frozenset**
frozenset is an immutable set
after creating frozenset we cannot add or remove items/objects
frozenset is immutable and all immutable are hashables
we can use frozenset as a nested set

How to create frozenset?
frozenset data type or class is used to represent frozenset object
**Syntax-1:** frozenset() ➔ cerating empty frozenset
**Syntax-2:** frozenset(iterable) ➔ creating frozenset using existing iterables

```
>>> fset1=frozenset()
>>> fset1.add(10)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    fset1.add(10)
AttributeError: 'frozenset' object has no attribute 'add'
>>>
>>> fset2=frozenset([10,20,30,40,50])
>>> print(fset2)
frozenset({40, 10, 50, 20, 30})
>>> fset3=frozenset({10,20,30,40,50})
>>> print(fset3)
frozenset({50, 20, 40, 10, 30})
>>> print(type(fset2),type(fset3))
<class 'frozenset'> <class 'frozenset'>
>>>
```

**Example:**

```
set1={frozenset({10,20,30}),frozenset({40,50,60})}
print(set1)
set2={frozenset({10,20,30}),frozenset({10,20,30})}
print(set2)
set3={frozenset({10,20,30}),frozenset({10,20,40})}
print(set3)
```

**Output:**

```
{frozenset({10, 20, 30}), frozenset({40, 50, 60})}
{frozenset({10, 20, 30})}
{frozenset({10, 20, 30}), frozenset({40, 10, 20})}
>>>
```

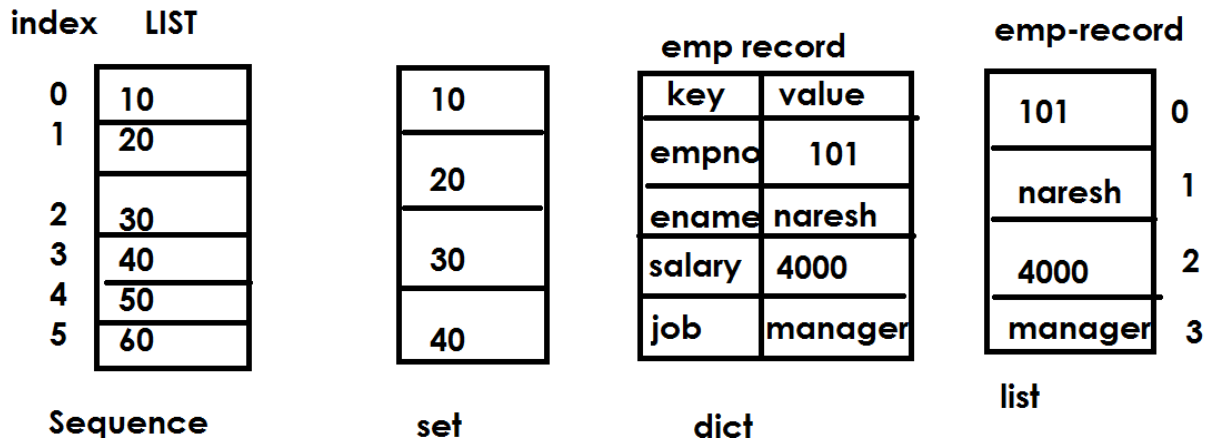
**Q: What is difference between set and frozenset?**

| **Set** | **Frozenset** |
|---|---|
| It is mutable collection | It is immutable collection |
| It is not hashable | It is hashable |
| It cannot be used to represent inner set | It can be used to represent inner set |
| Set uses {} for creating | Frozenset does not uses any symbol for creation |

**Mapping collection or dictionary or dict**

Dictionary is a key based collection
Dictionary is mapping collection
In mapping collection each key is mapping with one or more than one value
In dictionary data is organized as pair of values (key,value)
Duplicate keys are not allowed but duplicate values are allowed
In application development in order to organize data as key and value pair then uses dictionaries

A mapping object maps hashable values to arbitrary objects. Mappings are mutable objects. There is currently only one standard mapping type, the *dictionary*




**How to create dictionary?**

Dictionaries can be created by several means:

- Use a comma-separated list of key: value pairs within braces: {'jack': 4098, 'sjoerd': 4127} or {4098: 'jack', 4127: 'sjoerd'}
- Use a dict comprehension: {}, {x: x ** 2 for x in range(10)}
- Use the type constructor: dict(), dict([('foo', 100), ('bar', 200)]), dict(foo=100, bar=200)

**Example:**

```
>>> d1={}
>>> print(type(d1),d1)
<class 'dict'> {}
>>> d2={10}
>>> print(type(d2),d2)
<class 'set'> {10}
>>> d3={1:10,2:20,3:30}
>>> print(d3,type(d3))
{1: 10, 2: 20, 3: 30} <class 'dict'>
>>> d4={1:10,1:20,1:30}
>>> print(d4)
{1: 30}
>>>
```