
Python
Bootcamp
& Masterclass
frozensets
gknxt

A frozenset is an unordered collection of distinct, hashable objects
A frozenset is immutable, so it can be a member of a set/frozenset

- A frozenset is an unordered collection of unique and hashable objects. (So, lists and dictionaries cannot be members of a frozenset. A tuple can be a member if and only if all its members are immutable)

- A frozenset is immutable and hashable.

- A frozenset has to be created using the `frozenset()` function.

```
s1 = frozenset(['Hi', 3, 1, 3, 2])                    # frozenset function takes any iterable
print("object s1:", s1, "is of type:", type(s1))
```

```
object s1: frozenset({1, 2, 3, 'Hi'}) is of type: <class 'frozenset'>
```

```
s2 = frozenset()                                      # empty frozenset - cannot be modified
print("object s2:", s2, "is of type:", type(s2))
```

```
object s2: frozenset() is of type: <class 'frozenset'>
```

```
s3 = frozenset((5,))                                  # singleton frozenset
print("object s3:", s3, "is of type:", type(s3), "and of length:", len(s3))
```

```
object s3: frozenset({5}) is of type: <class 'frozenset'> and of length: 1
```

```
fs = frozenset(['Hi', 3, 1, 3, 2])
s4 = {4, True, ('a', 'b'), fs, 3.14}                   # set can contain frozenset
s4
```

```
{('a', 'b'), 3.14, 4, True, frozenset({1, 2, 3, 'Hi'})}
```

elements are unique

The elements of a frozenset must be unique, so if a frozenset is created with duplicate elements, all the duplicates will be removed. Since a frozenset is immutable, adding/deleting/updating elements is not allowed.

```
email_list = ['hr@gknxt.com', 'ai@gknxt.com', 'hr@gknxt.com', 'gk@gknxt.com']   # 'hr@gknxt.com' is a duplicate
email_set = frozenset(email_list)       # converts list into a set - removes duplicates, if present
email_set
```

```
frozenset({'ai@gknxt.com', 'gk@gknxt.com', 'hr@gknxt.com'})
```

```
email_list = ['hr@gknxt.com', 'ai@gknxt.com', 'hr@gknxt.com', 'gk@gknxt.com']
email_set = frozenset(email_list)
email_set.add('gk@gknxt.com')           # AttributeError: frozenset is immutable, cannot add elements to it
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8324/2886107259.py in <module>
      1 email_list = ['hr@gknxt.com', 'ai@gknxt.com', 'hr@gknxt.com', 'gk@gknxt.com']
      2 email_set = frozenset(email_list)
----> 3 email_set.add('gk@gknxt.com')           # AttributeError: frozenset is immutable, cannot add elements to it

AttributeError: 'frozenset' object has no attribute 'add'
```

gknxt

deleting all elements

The only way to remove all the elements of a frozenset is to make it equal to an empty set / frozenset, so a new object with the same identifier will be created.

`del` keyword deletes the entire frozenset.

```
states = ('NY', 'AZ', 'CA')
a = frozenset(states)
print("object a:", a, "has ID:", hex(id(a)), "is of type:", type(a))
a = frozenset()    # new object will be created with the same identifier
print("object ab:", a, "has ID:", hex(id(a)), "is of type:", type(a))
```

```
object a: frozenset({'AZ', 'CA', 'NY'}) has ID: 0x1c982b2bba0 is of type: <class 'frozenset'>
object ab: frozenset() has ID: 0x1c9fe4923c0 is of type: <class 'frozenset'>
```

```
states = ('NY', 'AZ', 'CA')
b = frozenset(states)
b = set()    # new object will be created with the same identifier
b
```

```
set()
```

```
states = ('NY', 'AZ', 'CA')
c = frozenset(states)
del c
```

string to a frozenset

A string is a sequence type, whose elements are simply its individual characters. Since the `frozenset()` takes a sequence and converts its elements to its set items, passing a string to the `frozenset()` creates a set of the string's individual characters after removing any duplicates.

```
a = frozenset('Hello!')
a
```

```
frozenset({'!', 'H', 'e', 'l', 'o'})
```

```
s = 'Mississippi'
list(s)
frozenset(s)
```

```
['M', 'i', 's', 's', 'i', 's', 's', 'i', 'p', 'p', 'i']
```

```
frozenset({'M', 'i', 'p', 's'})
```

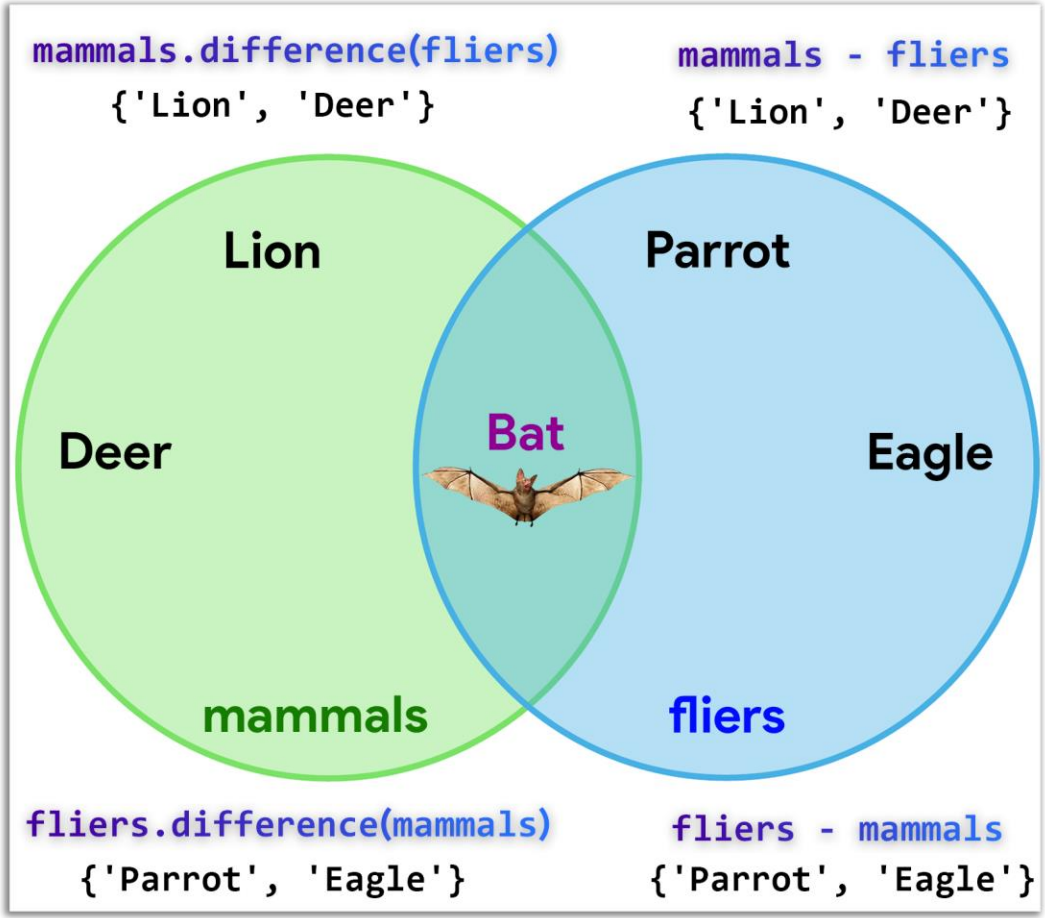
```
len(frozenset('A thing of beauty is a joy forever'))     # No. of unique chars in the quote
```

```
18
```

difference

- `difference(*others)` returns a new frozenset with elements in the frozenset that are not in the `others`

- The overloaded minus operator (`-`) can be used in place of `difference(*others)`



```
mammals = frozenset(["Lion", "Deer", "Bat"])
fliers = frozenset(["Parrot", "Eagle", "Bat"])
print("mammals.difference(fliers) =", mammals.difference(fliers))
print("mammals - fliers =", mammals - fliers)
print("fliers.difference(mammals) =", fliers.difference(mammals))
print("fliers - mammals =", fliers - mammals)
```

```
mammals.difference(fliers) = frozenset({'Deer', 'Lion'})
mammals - fliers = frozenset({'Deer', 'Lion'})
fliers.difference(mammals) = frozenset({'Eagle', 'Parrot'})
fliers - mammals = frozenset({'Eagle', 'Parrot'})
```

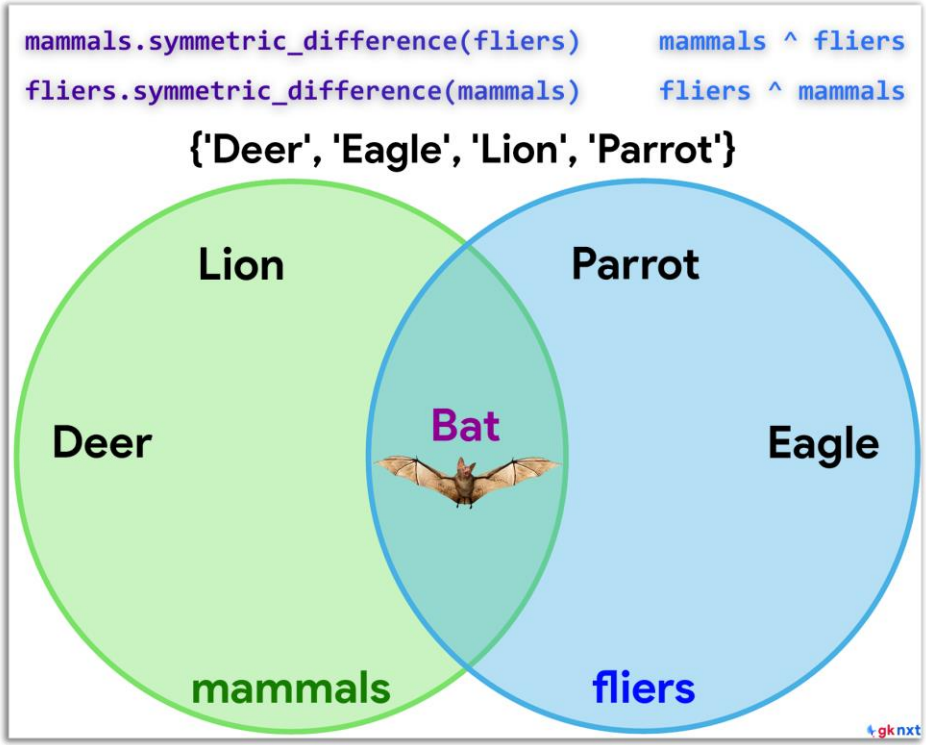
```
s = frozenset([1, 2, 3])
t = frozenset((1, 2))
s.difference(t)
s - t
t.difference(s)
t - s
```

```
frozenset({3})

frozenset({3})

frozenset()

frozenset()
```

symmetric difference

- `symmetric_difference(other)` returns a new frozenset with elements in either the frozenset or `other` but not both.

- Unlike `symmetric_difference`, overloaded **bitwise XOR operator** (`^`) can be used on multiple frozensets



gknxt

```
mammals = frozenset(["Lion", "Deer", "Bat"])
fliers = frozenset (["Parrot", "Eagle", "Bat"])
mammals.symmetric_difference(fliers)
fliers.symmetric_difference(mammals)
mammals ^ fliers
fliers ^ mammals
```

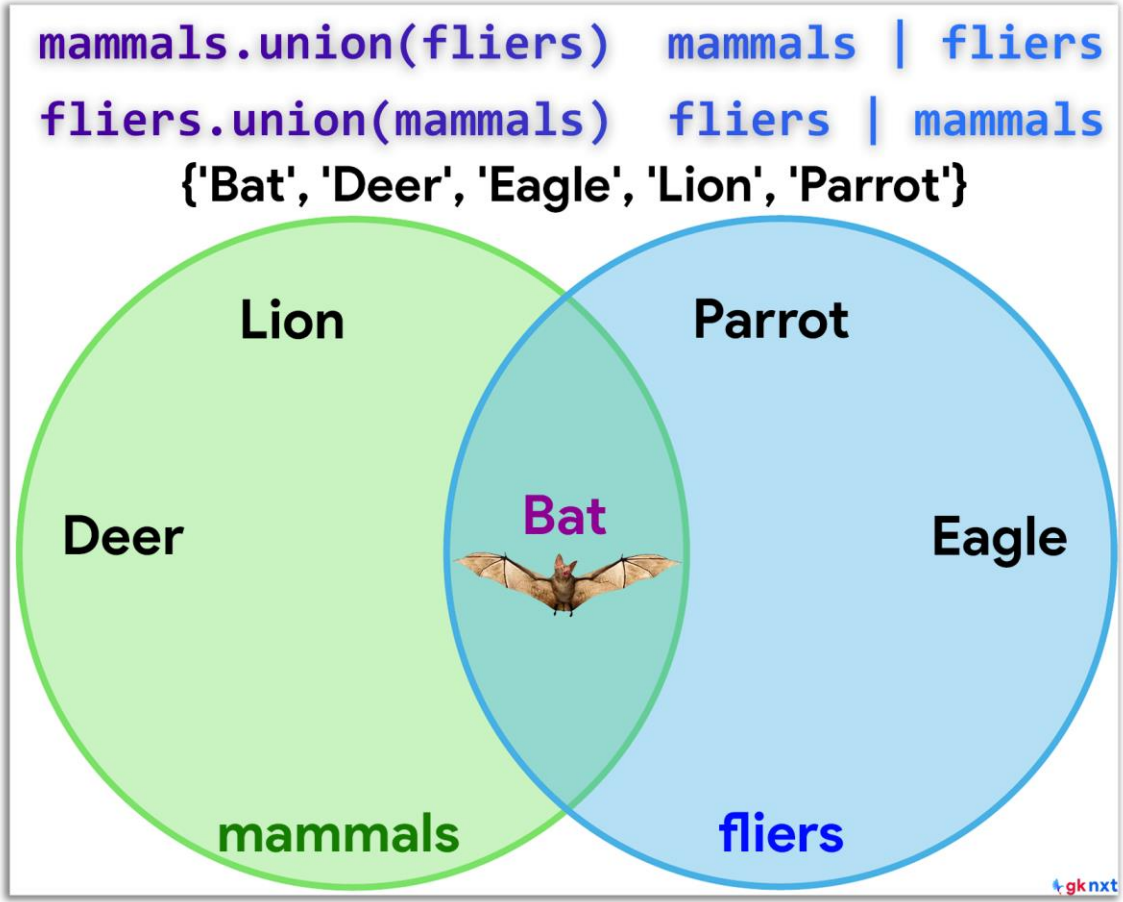
```
frozenset({'Deer', 'Eagle', 'Lion', 'Parrot'})

frozenset({'Deer', 'Eagle', 'Lion', 'Parrot'})

frozenset({'Deer', 'Eagle', 'Lion', 'Parrot'})

frozenset({'Deer', 'Eagle', 'Lion', 'Parrot'})
```

```
s = frozenset([1, 2, 3])
t = frozenset([2, 3, 4])
s.symmetric_difference(t)
t.symmetric_difference(s)
s ^ t
t ^ s
```

```
frozenset({1, 4})

frozenset({1, 4})

frozenset({1, 4})

frozenset({1, 4})
```

union

- `union(*others)` returns a new frozenset with elements from the frozenset and all `others`
- The overloaded **bitwise OR operator** (|) can be used in place of the `union(*others)`



gknxt

```
mammals = frozenset(["Lion", "Deer", "Bat"])
fliers = frozenset (["Parrot", "Eagle", "Bat"])
mammals.union(fliers)
fliers.union(mammals)
mammals | fliers
fliers | mammals
```

```
frozenset({'Bat', 'Deer', 'Eagle', 'Lion', 'Parrot'})

frozenset({'Bat', 'Deer', 'Eagle', 'Lion', 'Parrot'})

frozenset({'Bat', 'Deer', 'Eagle', 'Lion', 'Parrot'})

frozenset({'Bat', 'Deer', 'Eagle', 'Lion', 'Parrot'})
```

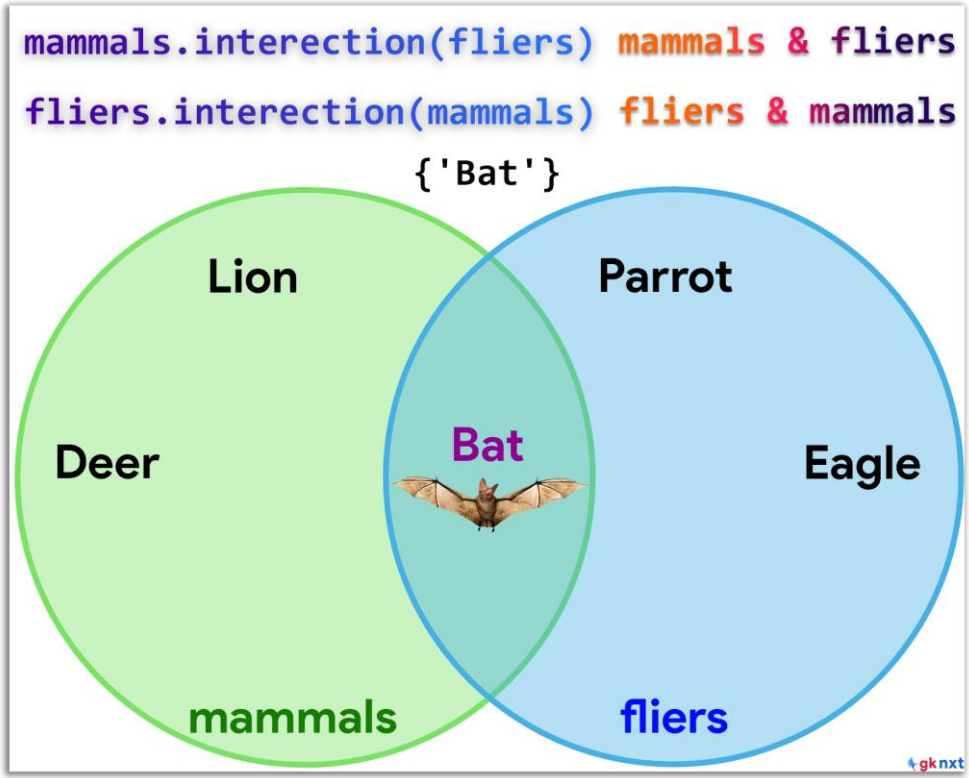
```
s = frozenset([1, 2, 3, 4])
t = frozenset([3, 4, 5])
s.union(t)
t.union(s)
s | t
t | s
```

```
frozenset({1, 2, 3, 4, 5})

frozenset({1, 2, 3, 4, 5})

frozenset({1, 2, 3, 4, 5})

frozenset({1, 2, 3, 4, 5})
```

intersection

- `intersection(*others)` returns a new frozenset with elements common to the frozenset and all `others`

- The overloaded **bitwise AND operator** (`&`) can be used in place of the `intersection(*others)`



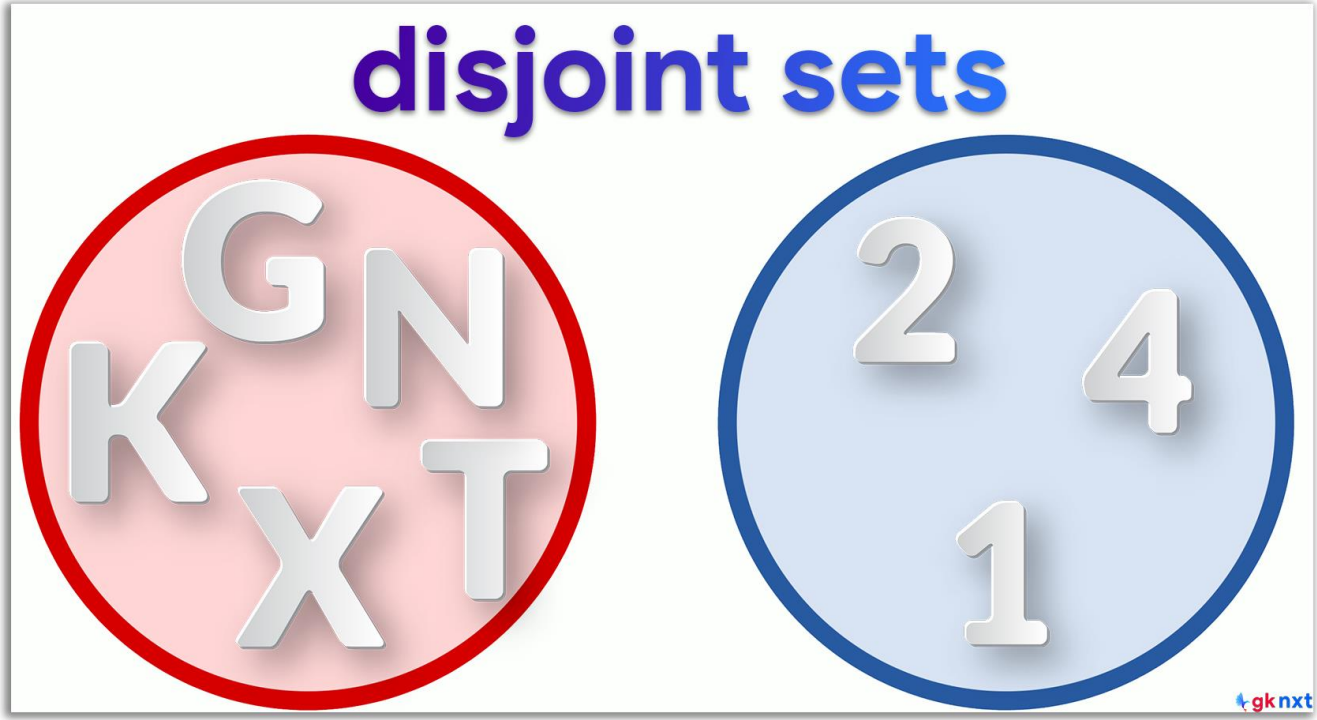
gknxt

```
mammals = frozenset(["Lion", "Deer", "Bat"])
fliers = frozenset (["Parrot", "Eagle", "Bat"])
mammals.intersection(fliers)
fliers.intersection(mammals)
mammals & fliers
fliers & mammals
```

```
frozenset({'Bat'})

frozenset({'Bat'})

frozenset({'Bat'})

frozenset({'Bat'})
```

```
s = frozenset([1, 2, 3, 4])
t = frozenset([3, 4, 5])
s.intersection(t)
t.intersection(s)
s & t
t & s
```

```
frozenset({3, 4})

frozenset({3, 4})

frozenset({3, 4})

frozenset({3, 4})
```



`isdisjoint(other)` returns `True` if the frozenset has no elements in common with other. Sets are disjoint if and only if their intersection is the empty set. There is no operator that corresponds to the .isdisjoint() method.

```
s = frozenset([4, 2, 1])
t = frozenset(['G', 'K', 'N', 'X', 'T'])
s.isdisjoint(t)
s.isdisjoint(t)
```

```
True

True
```

```
a = frozenset([9, 19, 29, 39, 49, 59])
b = frozenset([2, 5, 7, 11, 13, 17, 19])
a.isdisjoint(b)
b.isdisjoint(a)
```

```
False

False
```

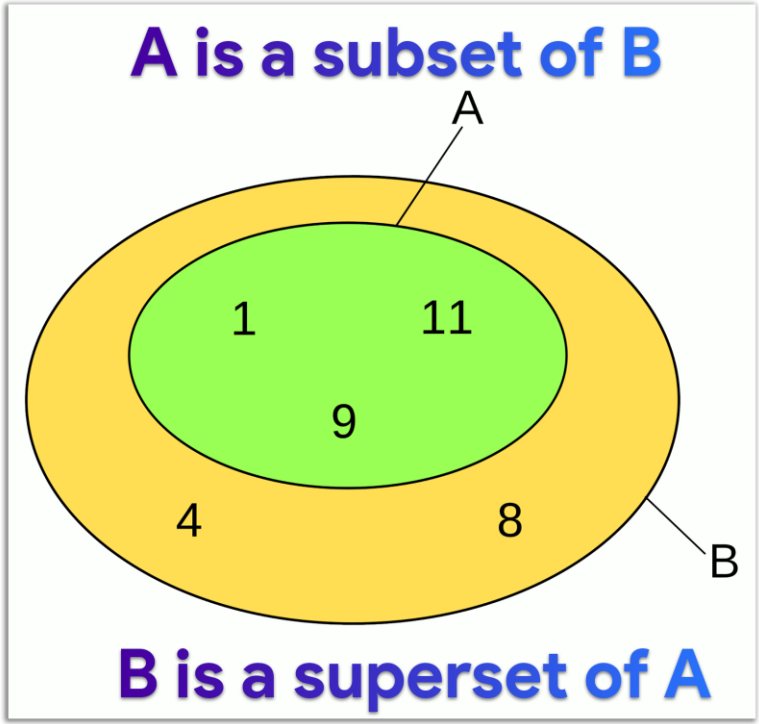
```
c = frozenset()
d = frozenset()
c.isdisjoint(d)   # intersection between two empty frozensets is empty
d.isdisjoint(c)   # two empty frozensets are disjoint to each other
```

```
True

True
```

subset & superset

- `issubset(other)` returns `True` if every element in the frozenset is in `other`. The symbol <= corresponds to subset and < corresponds to proper subset.

- `issuperset(other)` returns `True` if every element in the `other` is in frozenset. The symbol >= corresponds to superset and > corresponds to proper superset.



```
s = frozenset(['Bush', 'Biden'])
t = frozenset(['Reagan', 'Bush', 'Obama', 'Trump', 'Biden'])
s.issubset(t)
s <= t          # s <= t is same as s.issubset(t)
t.issuperset(s)
t >= s          # t >= s is same as t.issuperset(s)
```

```
True

True

True

True
```

```
s = ['Bush', 'Biden']     # s is a list
t = frozenset(['Reagan', 'Bush', 'Obama', 'Trump', 'Biden'])
t.issubset(s)             # issubset method can take any itrable as argument
t.issuperset(s)           # issuperset method can take any itrable as argument
```

```
False

True
```

operator Vs non-operator

- The non-operator versions of `union()`, `intersection()`, `difference()`, `symmetric_difference()`, `issubset()`, and `issuperset()` methods will accept any iterable as an argument. In contrast, their operator based counterparts require their arguments to be sets/frozensets.

- The > operator is the only way to test whether a frozensets is a proper superset or not. There is no corresponding method.

- The < operator is the only way to test whether a frozensets is a proper subset or not. There is no corresponding method.

copy()

The `copy()` method returns a shallow copy of the frozenset. Deep copy doesn't make any sense for sets because sets/frozensets are only allowed to contain immutable objects.

```
s = frozenset([11, 2, 5, 4, 5, 8, (3, 7), 5, 2])
t = s.copy()       # deep copy doesn't make sense for frozensets because they hold immutable objects only
print("set s:", s, "has id:", hex(id(s)))
print("set t:", t, "has id:", hex(id(t)))
```

```
set s: frozenset({2, 4, 5, 8, (3, 7), 11}) has id: 0x1c982b55900
set t: frozenset({2, 4, 5, 8, (3, 7), 11}) has id: 0x1c982b55900
```

in & not in

The `in` and `not in` operators can be used to test whether a value is in a frozenset or not.

```
s = frozenset(['Reagan', 'Bush', 'Obama', 'Trump', 'Biden'])
'Obama' in s
'Hillary' not in s
```

```
True

True
```

```
s = frozenset(['Reagan', 'Bush', ('Obama', 'Trump'), 'Biden'])
t = 'Obama'
u = ('Obama', 'Trump')
t in s
u in s
```

```
False

True
```

sets & frozensets

- Instances of set can be compared to instances of frozenset based on their members.

- Binary operations that mix set instances with frozenset return the type of the first operand

```
set('abc') == frozenset('abc')
frozenset('abc') == set('abc')
```

```
True

True
```

```
set('abc') in frozenset('abc')
frozenset('abc') in set('abc')
```

```
False

False
```

Online Resources

For best python resources, please visit:

gknxt.com/python/

gknxt

Python
Bootcamp
& Masterclass

Thank You

for your Rating & Review

gknxt