Sets in Python

What we will learn?

- Defining a Set
- Set size and Membership
- Operations on Set

Defining a Set

Mathematical Definition:

a set is a well-defined collection of **distinct** objects, typically called **elements** or **members**.

Python's built-in data type 'Set' has following characteristics:

- Sets are unordered
- Set elements are unique. Duplicate elements are not allowed
- A set itself is mutable i.e. you can add or remove elements from set
- Elements contained in the sets must be of **immutable type**. (strings, tuples, Numeric)
- Sets can have different type of elements that is they can be heterogeneous in nature

Creating a Set

A set can be created using two ways:

1. Built-in set ( ) function: It uses iterable
2. With curly braces { }: Make group of immutable elements

1. Built-in set( ) function

It can be defined as

```
x = set( <iter> )
```

Here, argument is an iterable. It could be of type 'list', 'tuple' or 'string'

x contains set of elements of iterable.

Resulting sets are unordered i.e. not necessarily in order in the definition statement

Duplicate elements are only represented only once

```
# List as an iterable
x = set ( ['Delhi', 'Mumbai', 'Kolkata', 'Chennai'] )
print (x)

# Tuple as an iterable
y = set( ('Delhi', 'London', 'New York', "Mumbai") )
print(y)

# String as an iterable. It will generate set of characters
z = set('Cape Town')
print(z)

# The below statement will throw error as number is not iterable
#m = set(10)

# The below statement also throws error as it accepts only one iterable
#n = set([1, 2, 3],[2.1, 3.2, 5.6])
```

```
{'Kolkata', 'Mumbai', 'Chennai', 'Delhi'}
{'Mumbai', 'London', 'New York', 'Delhi'}
{'o', 'n', 'T', ' ', 'w', 'p', 'C', 'e', 'a'}
```

2. Using Curly Braces

Sets can be defined using curly braces as follows:

```
x = { <obj1>, <obj2>, <obj3> ... <objn> }
```

Here each object becomes distinct element of the set.

These objects can be iterable as well.

```
x = {3, 5, 2.5, 7, 3}
print('x =',x)

# Example of Heterogeneous Set
y = { 'Cat', 'Tiger', 1.25, (2,3,5)}
print('y =',y)
```

```
x = {2.5, 3, 5, 7}
y = {1.25, (2, 3, 5), 'Cat', 'Tiger'}
```

Note: The objects placed in curly brackets in set remains intact even if they are iterable.

Check the difference

```
z = set('abc')
print(z)
```

```
m = {'abc'}
print(m)

     {'c', 'b', 'a'}
     {'abc'}
```

A set can be empty. We can use following declaration for it.

```
 x = set()
```

Empty set can't be declared with pair of curly braces as it is considered as empty dictionary

```
x = set()
print(x)

y = { }
print(y)

print('type of x:',type(x))
print('type of y:',type(y))

     set()
     {}
     type of x: <class 'set'>
     type of y: <class 'dict'>
```

Note: Elements in set must be of immutable type.

Mutable objects can't be element of set.

Hence list and dictionary elements could not be part of the set

```
a = [1, 2, 3]
# Below statement will throw the error
# st = {a}

k = { 'a':2, 'b':4 }
# Below statement will throw the error
# st = {k}
```

Set size and Membership

The **len( )** function returns number of elements in set.

The operator **'in'** and **'not in'** can be used to test membership of an element in set

```
st = {'Gold', 'Silver', 'Platinum', 'Bronze', 'Chromium'}
print("The given set is:",st)

# To get the length of set
l = len(st)
print("Number of elements in the set:",l)

s1 = 'Copper'

# To test the membership
if s1 in st:
  print(s1,'is in set')
else:
  print(s1,'is not in set')
```

```
The given set is: {'Silver', 'Gold', 'Bronze', 'Platinum', 'Chromium'}
Number of elements in the set: 5
Copper is not in set
```

Operations on Set

The sets in Python can be treated similar to Mathematical Sets.

All mathematical set operations can be performed.

Indexing and slicing can't be performed as there is no fixed ordering of elements.

Set operations can be performed in two ways:

1. By Operator
2. By Method

Operations to Generate new sets

Given sets x1 and x2, Possible operations on sets are

1. Union:

> Union operation combines elements of x1 and x2

> The method can be called as **x1.union(x2)**

> The operator (|) is also used for union operation as **x1 | x2**

2. Intersection:

> Intersection operation identifies common elements between x1 and x2

> The method can be called as **x1.intersection(x2)**

> The operator (&) is also used for union operation as **x1 & x2**

3. Difference:

> Difference operation identifies elements present in x1 but not in x2.

> The method can be called as **x1.difference(x2)**

> The operator (-) is also used for union operation as **x1 - x2**

4. Symmetric Difference:

> Symmetric Difference operation identifies elements present in x1 or in x2 but not in both.

> The method can be called as **x1.symmetric_difference(x2)**

> The operator (^) is also used for union operation as **x1 ^ x2**

Note:

1. For above operations, the resultant set has distinct elements
2. The method or operator does not change the participating sets.
3. These operations can be applied on more than one set as well

```
x1 = {'Delhi', 'Kolkata', 'Mumbai'}
x2 = {'Delhi', 'New York', 'London', "Berlin", "Mumbai"}

print('Set X1:',x1)
print('Set X2:',x2)

x3 = x1 | x2
x4 = x1.union(x2)
x5 = x1 & x2
x6 = x1.intersection(x2)
x7 = x1 - x2
x8 = x1.difference(x2)
x9 = x1 ^ x2
x10 = x1.symmetric_difference(x2)

print('Result of | operator:',x3)
print('Result of Union Method:',x4)
print('Result of & operator:',x5)
print('Result of Intersection Method:',x6)
print('Result of - operator:',x7)
print('Result of difference Method:',x8)
print('Result of ^ operator:',x9)
print('Result of Symmetric Difference Method:',x10)
```

```
Set X1: {'Kolkata', 'Mumbai', 'Delhi'}
Set X2: {'Mumbai', 'London', 'Berlin', 'New York', 'Delhi'}
Result of | operator: {'Mumbai', 'Kolkata', 'London', 'Berlin', 'New York', 'Delhi'}
Result of Union Method: {'Mumbai', 'Kolkata', 'London', 'Berlin', 'New York', 'Delhi'}
Result of & operator: {'Mumbai', 'Delhi'}
Result of Intersection Method: {'Mumbai', 'Delhi'}
Result of - operator: {'Kolkata'}
Result of difference Method: {'Kolkata'}
Result of ^ operator: {'Berlin', 'New York', 'Kolkata', 'London'}
Result of Symmetric Difference Method: {'Berlin', 'New York', 'Kolkata', 'London'}
```

Set operations on multiple sets

```
s1 = { 1, 2, 3, 4 }
s2 = { 2, 3, 4, 5 }
s3 = { 3, 4, 5, 6 }
s4 = { 4, 5, 6, 7 }

# Result of Union operation
s5 = s1 | s2 | s3 | s4
s6 = s1.union( s2, s3, s4 )
print("Union of All Sets using | operator:",s5)
print("Union of All Sets using method:",s6)

# Result of Intersection operation
s7 = s1 & s2 & s3 & s4
s8 = s1.intersection( s2, s3, s4 )
print("Intersection of All Sets using & operator:",s7)
print("Intersection of All Sets using method:",s8)

# Result of Difference operation
s9 = s1 - s2 - s3 - s4
s10 = s1.difference( s2, s3, s4 )
print("Difference of All Sets using - operator:",s9)
print("Difference of All Sets using method:",s10)

# Result of Symmetric Difference operation
s11 = s1 ^ s2 ^ s3 ^ s4

# The method for symmetric difference for multiple sets don't exist
#s12 = s1.symmetric_difference( s2, s3, s4 )
print("Symmetric Difference of All Sets using ^ operator:",s11)
```

```
Union of All Sets using | operator: {1, 2, 3, 4, 5, 6, 7}
Union of All Sets using method: {1, 2, 3, 4, 5, 6, 7}
Intersection of All Sets using & operator: {4}
Intersection of All Sets using method: {4}
Difference of All Sets using - operator: {1}
Difference of All Sets using method: {1}
Symmetric Difference of All Sets using ^ operator: {1, 3, 5, 7}
```

Note:

Whenever a set is expected, methods will typically accept any iterable as an argument. Then it will convert to set and complete the operation.

Operators require actual sets as operands strictly.

```
x1 = {'Delhi', 'Kolkata', 'Mumbai'}

# x2 is declared as tuple
x2 = ('Delhi', 'New York', 'London', "Berlin", "Mumbai")

x3 = x1.union(x2)
print("x3:",x3)

# The below statement will throw the error
# x4 = x1 | x2
```

```
x3: {'Mumbai', 'Kolkata', 'London', 'Berlin', 'New York', 'Delhi'}
```

Operations to determine relation between Sets

Given sets x1 and x2, relationship between sets can be determined. Possible relationships are as follows:

1. Disjoint:

> Whenever there is no common element between x1 and x2, the sets are called disjoint sets.

> The method can be called as **x1.isdisjoint(x2)**

> It returns 'True' if x1 and x2 are disjoint otherwise returns 'False'

> There is no operator defined to check disjoint relationship

2. Subset:

> It determines whether one set is subset of other.

> The method can be called as **x1.issubset(x2)**

> It returns 'True' if x1 is subset of x2 otherwise returns 'False'

> Less than or equal to operator (**<=**) can be used to check subset relationship

3. Proper Subset:

> It determines whether one set is proper subset of other.

> Less than operator (**<**) can be used to check proper subset relationship

> **x1 < x2** returns 'True' if x1 is proper subset of x2 otherwise returns 'False'

> There is no method dedicated for Proper Subset relationship

4. Superset:

> It determines whether one set is superset of other.

> The method can be called as **x1.issuperset(x2)**

> It returns 'True' if x1 is superset of x2 otherwise returns 'False'

> Greater than or equal to operator (**>=**) can be used to check subset relationship

5. Proper Superset:

> It determines whether one set is proper superset of other.

> Less than operator (**>**) can be used to check proper subset relationship

> **x1 > x2** returns 'True' if x1 is proper superset of x2 otherwise returns 'False'

> There is no method dedicated for Proper Superset relationship

Example for Relationship between sets

```
x1 = {'Delhi', 'Mumbai', 'London', 'Madrid', 'Melbourn'}
x2 = {'Delhi','Mumbai'}

x3 = x1 - x2
print('Set x3:',x3)

# Disjoint
t = x1.isdisjoint(x2)
print(t)

if (x2.isdisjoint(x3)):
  print('x2 and x3 are disjoint')
else:
  print('x2 and x3 are NOT disjoint')
  print("x2 and x3 are NOT disjoint")


# Subset and proper subset relationship
print("Set x1:",x1)
print("Set x2:",x2)
print("Set X3:",x3)

print("\nSubset Relationship")
t1 = x1.issubset(x2)
print("x1 is subset of x2:",t1)
t2 = x2.issubset(x1)
print("x2 is subset of x1:",t2)
t3 = x3 <= x1
print("x3 is subset of x1:",t3)
t4 = x3 <= x2
print("x3 is subset of x2:",t4)

print("\nProper Subset Relationship")
t1 = x1 < x2
print("x1 is proper subset of x2:",t1)
t2 = x2 < x2
print("x2 is proper subset of x2:",t2)
t3 = x3 < x1
print("x3 is proper subset of x1:",t3)
t4 = x3 < x2
print("x3 is proper subset of x2:",t4)


# Superset and Proper Superset Relationship
print()
print("Set x1:",x1)
print("Set x2:",x2)
print("Set X3:",x3)

print("\nSuperset Relationship")
t1 = x1.issuperset(x2)
print("x1 is superset of x2:",t1)
t2 = x2.issuperset(x1)
print("x2 is superset of x1:",t2)
t3 = x3 >= x1
print("x3 is superset of x1:",t3)
t4 = x3 >= x2
print("x3 is superset of x2:",t4)

print("\nProper superset Relationship")
t1 = x1 > x2
print("x1 is proper superset of x2:",t1)
t2 = x2 < x2
print("x2 is proper superset of x2:",t2)
t3 = x3 > x1
print("x3 is proper superset of x1:",t3)
```

```
t4 = x3 > x2
print("x3 is proper superset of x2:",t4)
```

```
Set x3: {'Madrid', 'Melbourn', 'London'}
False
x2 and x3 are disjoint
Set x1: {'Mumbai', 'London', 'Melbourn', 'Madrid', 'Delhi'}
Set x2: {'Mumbai', 'Delhi'}
Set X3: {'Madrid', 'Melbourn', 'London'}

Subset Relationship
x1 is subset of x2: False
x2 is subset of x1: True
x3 is subset of x1: True
x3 is subset of x2: False

Proper Subset Relationship
x1 is proper subset of x2: False
x2 is proper subset of x2: False
x3 is proper subset of x1: True
x3 is proper subset of x2: False

Set x1: {'Mumbai', 'London', 'Melbourn', 'Madrid', 'Delhi'}
Set x2: {'Mumbai', 'Delhi'}
Set X3: {'Madrid', 'Melbourn', 'London'}

Superset Relationship
x1 is superset of x2: True
x2 is superset of x1: False
x3 is superset of x1: False
x3 is superset of x2: False

Proper superset Relationship
x1 is proper superset of x2: True
x2 is proper superset of x2: False
x3 is proper superset of x1: False
x3 is proper superset of x2: False
```

```
# Function to check superset relationship

def check_relation(a,b):
  if a.issuperset(b):
    if a > b:
      print(a," is proper superset of ", b)
    else:
      print(a," is superset of ", b)
  else:
    print(a," is not superset of ", b)

check_relation(x1,x2)
check_relation(x2,x2)
check_relation(x3,x1)
```

```
{'Mumbai', 'London', 'Melbourn', 'Madrid', 'Delhi'}  is proper superset of  {'Mumbai',
{'Mumbai', 'Delhi'}  is superset of  {'Mumbai', 'Delhi'}
{'Madrid', 'Melbourn', 'London'}  is not superset of  {'Mumbai', 'London', 'Melbourn',
```

Modifying a set

Each of the union, intersection, difference, and symmetric difference operators has an augmented assignment form that can be used to modify a set.

Given sets x1 and x2, these operators can be explained as follows:

1. Update:

> Modifies set by Union

> Update operation add elements of x2 into x1 if it is not already present

> The method can be called as **x1.update(x2)**

> The operator (|=) is also used for update operation as **x1 |= x2**

2. Intersection Update:

> Modifies set by Intersection

> It retains elements in x1 if it is present in both x1 and x2

> The method can be called as **x1.intersection_update(x2)**

> The operator (&=) is also used for intersection update operation as **x1 &= x2**

3. Difference Update:

> Modifies set by Difference

> It removes elements from x1 if it is present in x2

> The method can be called as **x1.difference_update(x2)**

> The operator (-=) is also used for intersection update operation as **x1 -= x2**

4. Symmetric Difference Update:

> Modifies set by Symmetric Difference

> It retains elements of x1 if it is present in either x1 or x2 but not in both

> The method can be called as **x1.symmetric_difference_update(x2)**

> The operator (^=) is also used for intersection update operation as **x1 ^= x2**

```
x1 = {'Delhi', 'Kolkata', 'Mumbai'}
x2 = {'Delhi', 'New York', 'London', "Berlin", "Mumbai"}

print('Set X1:',x1)
print('Set X2:',x2)

x1 |= x2
print('Result of |= operator:',x1)

x1 = {'Delhi', 'Kolkata', 'Mumbai'}
x2 = {'Delhi', 'New York', 'London', "Berlin", "Mumbai"}

x1 &= x2
print('Result of &= operator:',x1)

x1 = {'Delhi', 'Kolkata', 'Mumbai'}
x2 = {'Delhi', 'New York', 'London', "Berlin", "Mumbai"}

x1 -= x2
print('Result of -= operator:',x1)

x1 = {'Delhi', 'Kolkata', 'Mumbai'}
x2 = {'Delhi', 'New York', 'London', "Berlin", "Mumbai"}

x1 ^= x2
print('Result of ^= operator:',x1)
```

```
Set X1: {'Kolkata', 'Mumbai', 'Delhi'}
Set X2: {'Mumbai', 'London', 'Berlin', 'New York', 'Delhi'}
Result of |= operator: {'Mumbai', 'Kolkata', 'London', 'Berlin', 'New York', 'Delhi'}
Result of &= operator: {'Mumbai', 'Delhi'}
Result of -= operator: {'Kolkata'}
Result of ^= operator: {'Kolkata', 'London', 'Berlin', 'New York'}
```

```
s1 = { 1, 2, 3, 4 }
s2 = { 2, 3, 4, 5 }

s1.update(s2)
print('Result of update method:',s1)

s1.intersection_update(s2)
print('Result of intersection_update method:',s1)

s1.difference_update(s2)
print('Result of difference_update method:',s1)
```

```
s1.symmetric_difference_update(s2)
print('Result of symmetric_difference_update method:',s1)
```

```
Result of update method: {1, 2, 3, 4, 5}
Result of intersection_update method: {2, 3, 4, 5}
Result of difference_update method: set()
Result of symmetric_difference_update method: {2, 3, 4, 5}
```

Other Methods for Modifying Set

1. x.add(element)

> The method adds an element to set.

> Element must be single immutable object

2. x.remove(element)

> The method removes an element from set.

> If element is not present then it raises an exception

3. x.discard(element)

> The method removes an element from set.

> If element is not present then it does not raise any exception

4. x.pop( )

> The method removes **random** element from the set

> If set is empty then it raises an exception

5. x.clear( )

> The method removes all elements from set

```
x = {'Delhi', 'New York', 'London', "Berlin", "Mumbai"}

x.add('Sydney')
print("Add Operation:",x)

x.remove('New York')
print("Remove Operation:",x)

# Below statement will raise exception
# x.remove('New York')
```

```
x.discard('London')
print("Discard Operation:",x)

# Although Key is not present there will not be any exception
x.discard('London')
print("Discard Operation:",x)

x.pop( )
print("Pop Operation:",x)

x.pop( )
print("Pop Operation:",x)

x.clear( )
print("Clear Operation:",x)
```

```
Add Operation: {'Mumbai', 'Sydney', 'London', 'Berlin', 'New York', 'Delhi'}
Remove Operation: {'Mumbai', 'Sydney', 'London', 'Berlin', 'Delhi'}
Discard Operation: {'Mumbai', 'Sydney', 'Berlin', 'Delhi'}
Discard Operation: {'Mumbai', 'Sydney', 'Berlin', 'Delhi'}
Pop Operation: {'Sydney', 'Berlin', 'Delhi'}
Pop Operation: {'Berlin', 'Delhi'}
Clear Operation: set()
```

✓ 0s completed at 00:22