Set Basics, Properties & Operations

August 11, 2021

A set in general is a collection of distinct objects. This applies in Python too, where a set is a data type that contains a finite collection of distinct objects. The data types of the objects need not be homogenous, but duplicates of a value are removed, even if they present in different variables. Also, a set is mutable, but elements can only be removed, not appended or inserted.

1 Properties

```
[13]: mySet = {1, 3.5, "Jackal"}
      mySet
```

```
[13]: {1, 3.5, 'Jackal'}
```

1.1 No duplicates

```
[5]: mySet = {1, 2, 3, 3, 3, 6, 7, 8, 8}
     mySet
```

```
[5]: {1, 2, 3, 6, 7, 8}
```

```
[6]: a = 2
     b = 2
     mySet = {a, b}
     mySet
```

```
[6]: {2}
```

1.2 Mutable

Since a set is unordered, the elements of a set cannot be indexed, hence we cannot access and change individual elements of a set. However, a set is mutable, so we can add elements and remove elements from a set, either particular values or the last value. Here, we only discuss functions that actually change the original set, by adding or removing elements. Set operations do not necessarily add or remove elements from the original set, and may instead return a new separate set (we will discuss the updating versions of each set operation too).

1.2.1 Removing elements from a set

[4]:
```python
mySet = {1, 2, 3, 4, 5, 6}
# To simply remove an element
print("Simply removing an element...")
mySet.remove(1)
print(mySet)
# To remove and return the last element of a set
print("Popping an element...")
print("Popped value:", mySet.pop())
print(mySet)
# Emptying the set completely
print("Emptying a set completely...")
mySet.clear()
print(mySet)
```

```
Simply removing an element…
{2, 3, 4, 5, 6}
Popping an element…
Popped value: 2
{3, 4, 5, 6}
Emptying a set completely…
set()
```

Note that mySet.discard(x) will achieve the same thing as mySet.remove(x).

1.2.2 Adding elements

[ ]:
```python
a = {"a", "b", "c", "d"}
```

[17]:
```python
a = {"a", "b", "c", "d"}
a.add(3)
a
```

[17]:
```
{3, 'a', 'b', 'c', 'd'}
```

Note that this function changes the original set. Also, you cannot use this to combine two sets.

[14]:
```python
a = {"a", "b", "c", "d"}
b = {1, 2, 3, 4}
a.add(b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-bbc3b730d6b5> in <module>
      1 a = {"a", "b", "c", "d"}
      2 b = {1, 2, 3, 4}
----> 3 a.add(b)
```

```
TypeError: unhashable type: 'set'
```

1.2.3 Adding sets

```
[1]: a = {"a", "b", "c", "d"}
     b = {1, 2, 3, 4}
     a.update(b)
     a
```

```
[1]: {1, 2, 3, 4, 'a', 'b', 'c', 'd'}
```

The argument of the update function can be any mutable collection data type. Hence, a tuple, if passed, will not have its elements considered and stored into the set. If you pass a dictionary, only the keys will be considered and stored.

```
[10]: a = {"w", "x", "y", "z"}
      b = [5, 6, 7, 8]
      d = (10, 11, 12, 13)
      d = {1 : "alpha", 2 : "beta"}
      a.update(b)
      a.update(c)
      a.update(d)
      a
```

```
[10]: {1, 2, 5, 6, 7, 8, 'w', 'x', 'y', 'z'}
```

```
[19]: # To demonstrate removal of duplicates yet again...
      a = {"a", "b", "c", "d"}
      b = {"a", "b", 1, 2, 3, 4}
      a.update(b)
      a
```

```
[19]: {1, 2, 3, 4, 'a', 'b', 'c', 'd'}
```

Note that this function changes the original set. Also, you cannot use this to add a single non-set element. This is the job of the add function.

1.3 No unhashable objects as elements

A hashable object is an object whose hash value does not change during its lifetime. A hash value is a numeric value of a fixed length that uniquely identifies data. In general, mutable containers such as lists and dictionaries are not hashable. Hash values represent large amounts of data as much smaller numeric values, by mapping large range of values (all possible instances of an class or data type) and map them onto a smaller set of values (such as a 128 bit number).

```
[22]: mySet = {1, 3.5, [1, 2, 3]}
      mySet
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-50d01379f969> in <module>
----> 1 mySet = {1, 3.5, [1, 2, 3]}
      2 mySet

TypeError: unhashable type: 'list'
```

```
[23]: mySet = {1, 3.5, ("a", "b", 5)}
      mySet
```

```
[23]: {('a', 'b', 5), 1, 3.5}
```

```
[24]: mySet = {1, 3.5, {"Hello", 5, 6}}
      mySet
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-11abff7e5199> in <module>
----> 1 mySet = {1, 3.5, {"Hello", 5, 6}}
      2 mySet

TypeError: unhashable type: 'set'
```

As we see above, out of list, tuple and set, only tuple objects are hashable, hence out of these three, only tuple objects an be single elements in a set. My best explanation is as follows… A tuple, once initialised, is of a fixed length, which means it is possible to map every variation of a tuple into a numeric value of fixed length. However, a list is not of fixed length, and hence, it is not possible or practical to map every possible every possible variation of a list into a numeric value of a fixed length. Even a set is not of fixed length, since you can add new elements and remove existing elements from a set.

1.4 Unordered

```
[7]: # Initalising a set
     mySet = {1, "Pranav", 2, 3,"Cat", 4.5, "Hello", 34}
     mySet
```

```
[7]: {1, 2, 3, 34, 4.5, 'Cat', 'Hello', 'Pranav'}
```

You can see above that a set is completely unordered, meaning that you cannot access a set's elements though indices, as you can in lists or tuples.

1.4.1 Non-subscriptable (cannot index a set)

As a consequence of a set being completely unordered, we cannot use indices to access elements of a set.

[27]:
```
mySet = {1, 2, 3, 4}
mySet[2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-884612ca441a> in <module>
      1 mySet = {1, 2, 3, 4}
----> 2 mySet[2]

TypeError: 'set' object is not subscriptable
```

2 Invalid operations

[34]:
```
a = {1, 2, 3, 4}
b = {"a", "b", "c", "d"}
```

[32]:
```
a + b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-48ef2860dc54> in <module>
----> 1 a + b
      2 2 * a

TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

[33]:
```
2 * a
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-829365422efa> in <module>
----> 1 2 * a

TypeError: unsupported operand type(s) for *: 'int' and 'set'
```

3 Checking for an element in a set

[46]:
```
mySet = {1, "Pranav", 2, 3,"Cat", 4.5, "Hello", 34}
print("My set:", mySet)
print("Is 4 in this set? ", 4 in mySet)
print("Is 'Hello' in this set? ", "Hello" in mySet)
```

```
My set: {1, 2, 3, 4.5, 34, 'Hello', 'Cat', 'Pranav'}
Is 4 in this set?  False
Is 'Hello' in this set?  True
```

4 Iterating through a set

[65]:
```
mySet = {1, "Pranav", 2, 3,"Cat", 4.5, "Hello", 34}
print("My set:", mySet)
for i in mySet:
    print(i)
```

```
My set: {1, 2, 3, 4.5, 34, 'Hello', 'Cat', 'Pranav'}
1
2
3
4.5
34
Hello
Cat
Pranav
```

5 Set operations

Note: in the function a.function(b), we term ‘a’ as the original set.

5.1 Union

[37]:
```
a = {1, 2, 3, 4, 7}
b = {3, 4, 5, 6, 7}
print(a.union(b))
print(a)
```

```
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 7}
```

The above function does not change the original set. To change the original set to the union’s result directly, we can use update function, with the same syntax.

5.2 Intersection

[29]:
```
a = {1, 2, 3, 4, 7}
b = {3, 4, 5, 6, 7}
a.intersection(b)
```

[29]:
```
{3, 4, 7}
```

The above function does not change the original set. To change the original set to the intersection's result directly, we can use the intersection_update function, with the same syntax.

The following operation is equivalent to the intersection function (no updation of any set).

[ ]:
```
a = {1, 2, 3, 4, 7}
b = {3, 4, 5, 6, 7}
a&b
```

5.3 Set difference

[28]:
```
a = {1, 2, 3, 4, 7}
b = {3, 4, 5, 6, 7}
a.difference(b)
```

[28]:
```
{1, 2}
```

This function only returns the values in the set 'a' that differ from the values in set 'b'.

The above function does not change the original set. To change the original set to the difference's result directly, we can use difference_update function, with the same syntax.

5.4 Symmetric difference

[35]:
```
a = {1, 2, 3, 4, 7}
b = {3, 4, 5, 6, 7}
a.symmetric_difference(b)
```

[35]:
```
{1, 2, 5, 6}
```

This function returns all the values, in both sets 'a' and 'b', that are not common. It is equivalent to a.difference(b).union(b.difference(a))

The above function does not change the original set. To change the original set to the symmetric difference's result directly, we can use symmetric_difference_update function, with the same syntax.