Python Sets

Set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable. However, the set itself is mutable (we can add or remove items). Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

Creating a Set in Python

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function `set()`. It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

```
>>> # set of integers
>>> my_set = {1, 2, 3}

>>> # set of mixed datatypes
>>> my_set = {1.0, "Hello", (1, 2, 3)}

>>> # set donot have duplicates
>>> {1,2,3,4,3,2}
{1, 2, 3, 4}

>>> # set cannot have mutable items
>>> my_set = {1, 2, [3, 4]}
...
TypeError: unhashable type: 'list'

>>> # but we can make set from a list
>>> set([1,2,3,2])
{1, 2, 3}
```

Creating an empty set is a bit tricky. Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the `set()` function without any argument.

```
>>> a = {}
>>> type(a)
<class 'dict'>
>>> a = set()
>>> type(a)
```

```
<class 'set'>
```

Changing a Set in Python

Sets are mutable. But since they are unordered, indexing have no meaning. We cannot access or change an element of set using indexing or slicing. Set does not support it. We can add single elements using the method `add()`. Multiple elements can be added using `update()` method. The `update()` method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
>>> my_set = {1,3}
>>> my_set[0]
...
TypeError: 'set' object does not support indexing
>>> my_set.add(2)
>>> my_set
{1, 2, 3}
>>> my_set.update([2,3,4])
>>> my_set
{1, 2, 3, 4}
>>> my_set.update([4,5], {1,6,8})
>>> my_set
{1, 2, 3, 4, 5, 6, 8}
```

Removing Elements from a Set

A particular item can be removed from set using methods like `discard()` and `remove()`. The only difference between the two is that, while using `discard()` if the item does not exist in the set, it remains unchanged. But `remove()` will raise an error in such condition. The following example will illustrate this.

```
>>> my_set = {1, 3, 4, 5, 6}
>>> my_set.discard(4)
>>> my_set
{1, 3, 5, 6}
>>> my_set.remove(6)
>>> my_set
{1, 3, 5}
>>> my_set.discard(2)
>>> my_set
{1, 3, 5}
>>> my_set.remove(2)
```

```
...
KeyError: 2
```

Similarly, we can remove and return an item using the `pop()` method. Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary. We can also remove all items from a set using `clear()`.

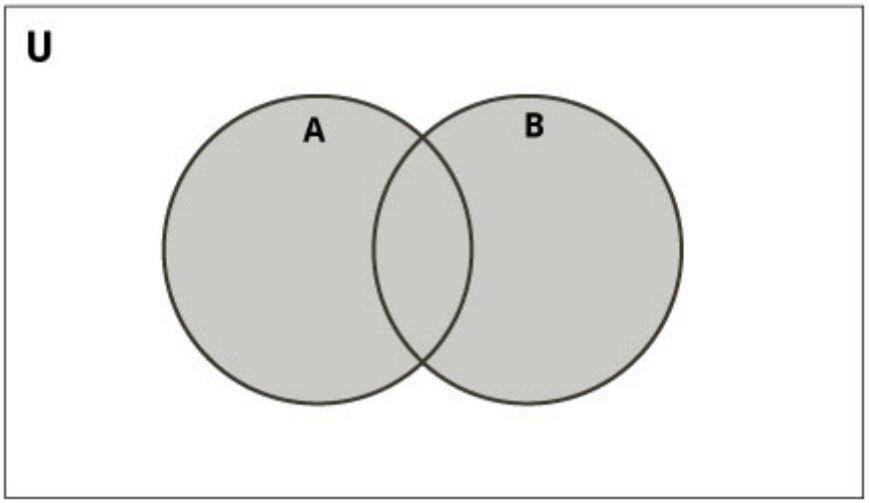
```
>>> my_set = set("HelloWorld")
>>> my_set.pop()
'r'
>>> my_set.pop()
'W'
>>> my_set
{'d', 'e', 'H', 'o', 'l'}
>>> my_set.clear()
>>> my_set
set()
```

Python Set Operation

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods. Let us consider the following two sets for the following operations.

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
```

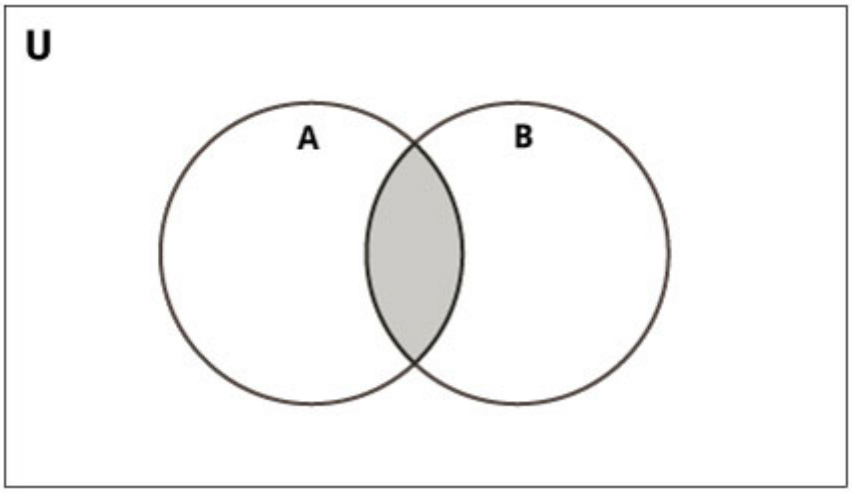
Set Union



Union of *A* and *B* is a set of all elements from both sets. Union is performed using `|` operator. Same can be accomplished using the method `union()`.

```
>>> A | B
{1, 2, 3, 4, 5, 6, 7, 8}
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

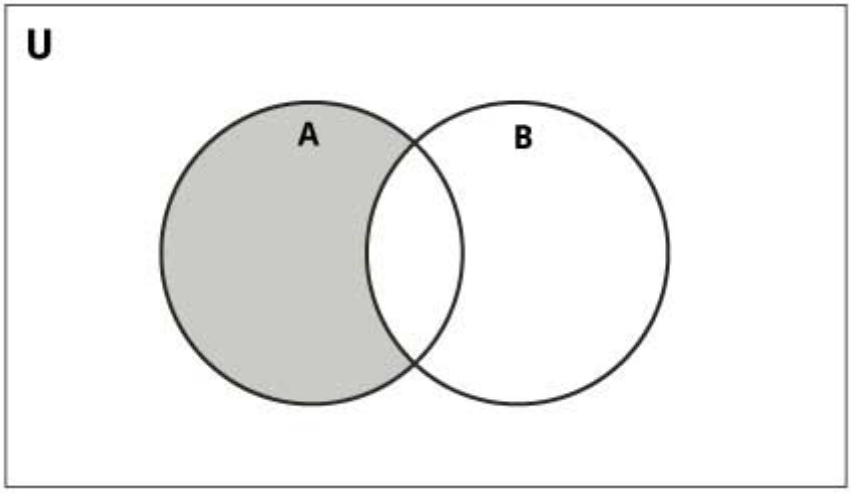
Set Intersection



Intersection of *A* and *B* is a set of elements that are common in both sets. Intersection is performed using `&` operator. Same can be accomplished using the method `intersection()`.

```
>>> A & B
{4, 5}
>>> A.intersection(B)
{4, 5}
>>> B.intersection(A)
{4, 5}
```

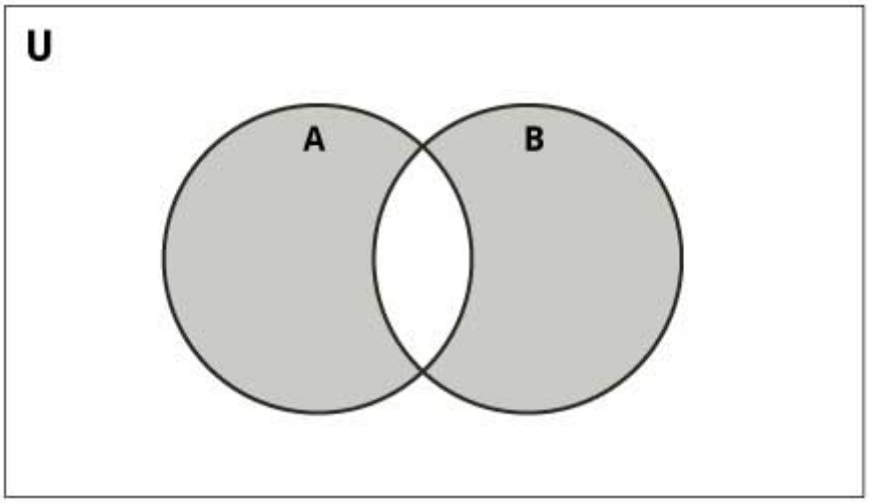
Set Difference



Difference of *A* and *B* (*A* - *B*) is a set of elements that are only in *A* but not in *B*. Similarly, *B* - *A* is a set of element in *B* but not in *A*. Difference is performed using `-` operator. Same can be accomplished using the method `difference()`.

```
>>> A - B
{1, 2, 3}
>>> A.difference(B)
{1, 2, 3}
>>> B - A
{8, 6, 7}
>>> B.difference(A)
{8, 6, 7}
```

Set Symmetric Difference



Symmetric Difference of *A* and *B* is a set of element in both *A* and *B* except those common in both. Symmetric difference is performed using `^` operator. Same can be accomplished using the method `symmetric_difference()`.

```
>>> A ^ B
{1, 2, 3, 6, 7, 8}
>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}
>>> B.symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```

Python Set Methods

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with set objects.

Python Set Methods	
Method	**Description**
add()	Add an element to a set
clear()	Remove all elemets form a set
copy()	Return a shallow copy of a set
difference()	Return the difference of two or more sets as a new set
difference_update()	Remove all elements of another set from this set
discard()	Remove an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Return the intersection of two sets as a new set
intersection_update()	Update the set with the intersection of itself and another
isdisjoint()	Return `True` if two sets have a null intersection
issubset()	Return `True` if another set contains this set
issuperset()	Return `True` if this set contains another set
pop()	Remove and return an arbitary set element. Raise `KeyError` if the set is empty
remove()	Remove an element from a set. It the element is not a member, raise a `KeyError`
symmetric_difference()	Return the symmetric difference of two sets as a new set

symmetric_difference_update()	Update a set with the symmetric difference of itself and another
union()	Return the union of sets in a new set
update()	Update a set with the union of itself and others

Other Set Operations

Set Membership Test

We can test if an item exists in a set or not, using the keyword `in`.

```
>>> my_set = set("apple")
>>> 'a' in my_set
True
>>> 'p' not in my_set
False
```

Iterating Through a Set

Using a for loop we can iterate though each item in a set.

```
>>> for letter in set("apple"):
...     print(letter)
...
a
p
e
l
```

Built-in Functions with Set

Built-in functions like `all()`, `any()`, `enumerate()`, `len()`, `max()`, `min()`, `sorted()`, `sum()` etc. are commonly used with set to perform different tasks.

Built-in Functions with Set	
Function	**Description**
all()	Return `True` if all elements of the set are true (or if the set is empty).
any()	Return `True` if any element of the set is true. If the set is empty, return `False.`

enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.
len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Retrun the sum of all elements in the set.

Python Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets. Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary.

Frozensets can be created using the function `frozenset()`. This datatype supports methods like `copy()`, `difference()`, `intersection()`, `isdisjoint()`, `issubset()`, `issuperset()`, `symmetric_difference()` and `union()`. Being immutable it does not have method that add or remove elements.

```
>>> A = frozenset([1, 2, 3, 4])
>>> B = frozenset([3, 4, 5, 6])
>>> A.isdisjoint(B)
False
>>> A.difference(B)
frozenset({1, 2})
>>> A | B
frozenset({1, 2, 3, 4, 5, 6})
>>> A.add(3)
...
AttributeError: 'frozenset' object has no attribute 'add'
```