SETS

Remember set theory? Well, these are the exact same sets. Sets in Python are a data structure, which is a collection of unordered and unindexed data. They can have exactly one instance of each data, so no duplicates.

To create a set, you can use the curly braces to enclose the comma separated data, or use the set constructor.

```
>>> {'Sun', 'Sat', 'Wed', 'Mon', 'Thu', 'Tue', 'Fri'}
{'Sat', 'Fri', 'Sun', 'Wed', 'Mon', 'Thu', 'Tue'}  # order is random

>>> set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])  # notice the square
brackets
{'Sat', 'Thu', 'Sun', 'Wed', 'Mon', 'Fri', 'Tue'}
```

Add and remove elements from set:

Sets have the `add()` and `discard()` methods. Since they don't have any duplicates, there is no point in providing index of the element to be added/removed.

```
>>> days = set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat"])
>>> days.add("Sun")
>>> print(days)
{'Wed', 'Fri', 'Tue', 'Mon', 'Sun', 'Thu', 'Sat'}
>>> days.discard("Mon")  # everybody hates Mondays anyway
>>> print(days)
{'Fri', 'Sat', 'Sun', 'Wed', 'Thu', 'Tue'}
```

The `update()` method can be used to add multiple elements.

```
>>> days.update(["PyDay", "AppleIsShitDay"])  # we will now follow a 9 day week
>>> print(days)
{'Sat', 'AppleIsShitDay', 'Thu', 'Tue', 'Wed', 'Sun', 'PyDay', 'Mon', 'Fri'}
```

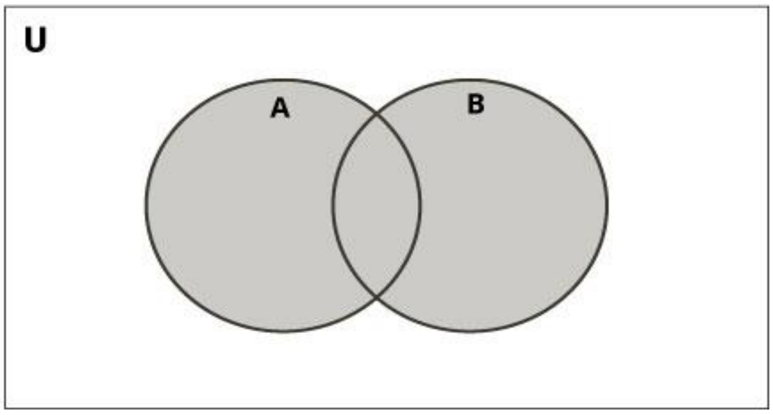
Operations on sets:

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}
```

1. Union

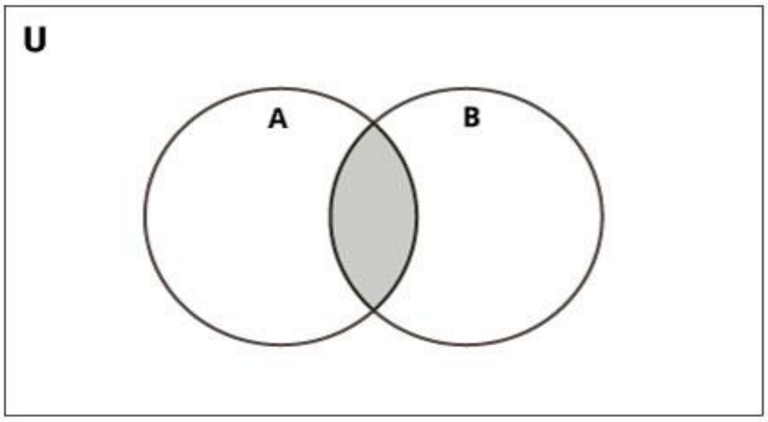


Union is performed using `|` operator. Same can be accomplished using the method `union()`.

```
>>> a | b
{1, 2, 3, 4, 5, 6, 7, 8}
```

```
>>> a.union(b)  # does not change a. A new set is returned
{1, 2, 3, 4, 5, 6, 7, 8}
```

2. Intersection

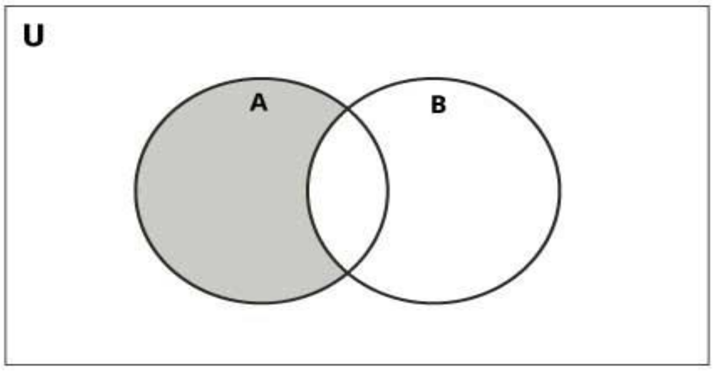


Intersection is performed using `&` operator. Same can be accomplished using the method `intersection()`.

```
>>> a & b
{4, 5}
```

```
>>> a.intersection(b)
{4, 5}
```

3. Difference



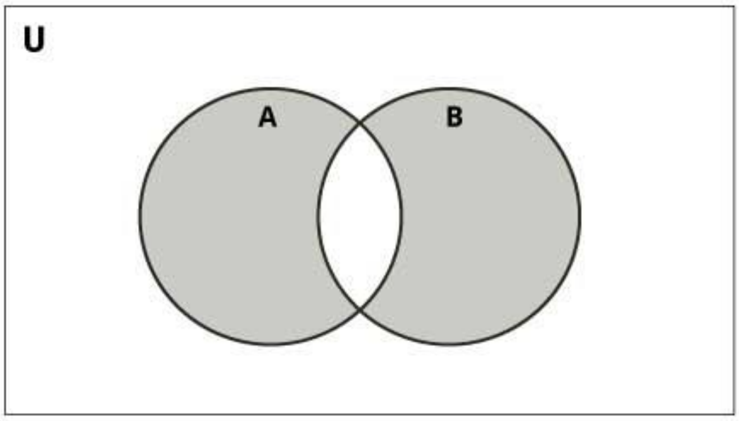
Difference of A and B (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A.
Difference is performed using – operator. Same can be accomplished using the method `difference()`.

```
>>> a - b
{1, 2, 3}

>>> b - a
{8, 6, 7}

>>> a.difference(b)
{1, 2, 3}
```

4. Symmetric difference



Symmetric Difference of A and B is a set of elements in both A and B except those that are 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}

>>> b ^ a
{1, 2, 3, 6, 7, 8}

>>> a.symmetric_difference(b)
{1, 2, 3, 6, 7, 8}
```

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.

Method	Description
add()	Add an element to a set
clear()	Remove all elements 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. If 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

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")  # initialize my_set
>>> print('a' in my_set)  # check if 'a' is present
True
>>> print('p' not in my_set)  # check if 'p' is not present
False
```

Comparisons

We can check if a given set is a subset or superset of another set. The result is True or False depending on the elements present in the sets.

```
>>> a = set(["Mon", "Tue", "Wed"])
>>> b = set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])
>>> a <= b  # subset
True
>>> b >= a  # superset
True
```