{ Python }

SETS

{ Collection Data Type }

Python Sets

We have learned lists, tuples, and strings (which are sequences). Elements of sequences are in order and can be accessed using an index. Also, these sequences can have duplicate elements.

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

Creating Sets

A set is created by placing all the items (elements) inside curly braces `{}`, separated by commas.

You can also use the built-in function `set()` to create sets.

```
# set of integers
my_set = {1, 2, 3}
print(my_set)   # Output: {1, 2, 3}

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)   # Output: {'Hello', 1.0,
(1, 2, 3)}
```

Creating Sets (Part II)

Let's try to create a set with duplicate elements.

```
my_set = {1, 2, 1, 3, 1, 2, 3}
print(my_set)      # Output: {1, 2, 3}
```

As you can see, `my_set` doesn't have any duplicate elements.

Now, let's create a set using the `set()` function.

```
my_set = set([1, 4, 1])
print(my_set)  # Output: {1, 4}
```

Here, we have passed a list to the `set()` function to create a set.

Creating empty set

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

```
# Empty set
my_set = set()

print(my_set)  # Output: set()
```

Add and Update Sets

Sets are mutable. But since they are unordered, indexing has no meaning.

We cannot access or change elements of a set using indexing or slicing. Set does not support it.

We can add a single element using the `add()` method and multiple elements using the `update()` method. The `update()` method can take tuples, lists, strings or other sets as its argument.

```
my_set = {1, 3}
print(my_set) # Output: {1, 3}

my_set.add(2)
print(my_set) # Output: {1, 2, 3}

my_set.update([4, 5, 6])
print(my_set) # Output: {1, 2, 3, 4, 5, 6}
```

The `update()` method can take tuples, strings and other sets as its argument.

Removing Elements from a Set

To remove elements from a set, we can either use `remove()` or `discard()` method.

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 conditions.

```
my_set = {1, 3, 4, 5, 6}

my_set.discard(4)
print(my_set)    # Output: {1, 3, 5, 6}

my_set.remove(6)
print(my_set)    # Output: {1, 3, 5}

# trying to remove 2 which doesn't exist
my_set.discard(2)
print(my_set)    # Output: {1, 3, 5}

my_set.remove(2)  # Error
```

Set Methods: pop() and clear()

The `pop()` method removes a random element from a set and returns it.

```
my_set = {'a', 'b', 'c'}
returned_value = my_set.pop()

print(my_set)    # Output: {'c', 'a'}
print(returned_value)  # Output: 'b'
```

Note: You may get a different result.

The `clear()` method removes all elements from a set.

```
my_set = {1, 2, 3}

my_set.clear()    # empty set
print(my_set)     # Output: set()
```

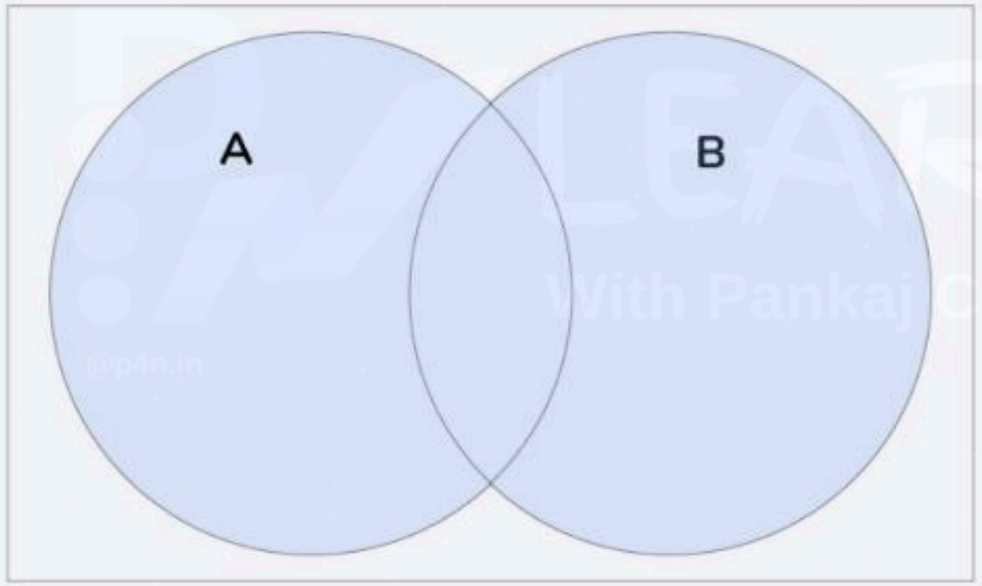
Python Set Operations

As you may have noticed, sets in Python are similar to sets in mathematics.

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.

Set Union

The union of two sets A and B is a set of all elements from both sets.



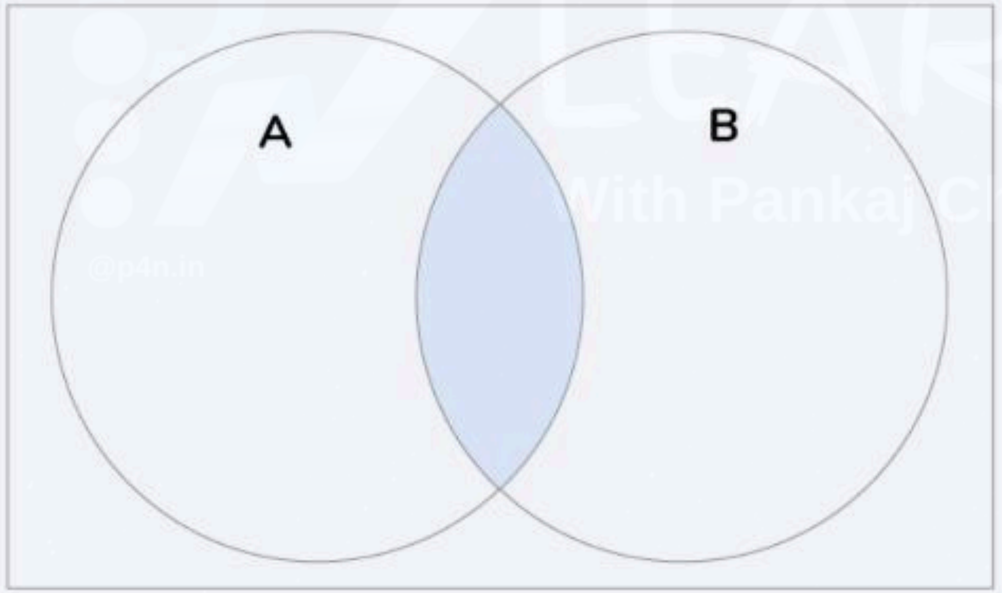
It is performed using | operator. Same can be accomplished using the `union()` method.

```
A = {1, 2, 3}
B = {2, 3, 4, 5}

result = A | B
print(result)    # {1, 2, 3, 4, 5}

result = B.union(A)
print(result)    # {1, 2, 3, 4, 5}
```

Note that, for two sets A and B, A union B is equal to B union A.

Set Intersection

The intersection of two sets A and B is a set of all elements that are common in both sets.



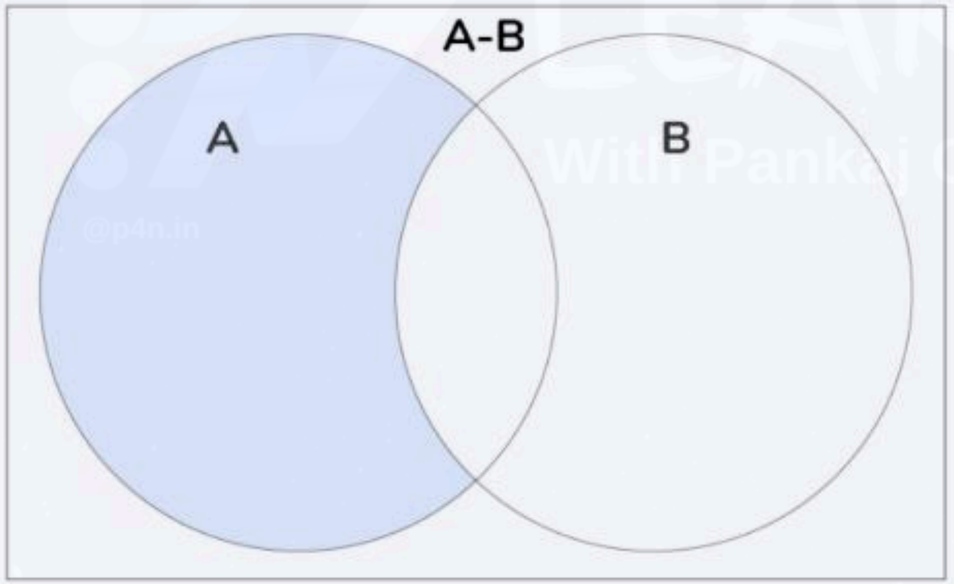
It is performed using & operator. Same can be accomplished using the method `intersection()`.

```
A = {1, 2, 3}
B = {2, 3, 4, 5}

result = A & B
print(result)    # {2, 3}

result = B.intersection(A)
print(result)    # {2, 3}
```

Note that, for two sets A and B, A intersection B is equal to B intersection A.

Set Difference

The difference of set `B` from `A` is a set of elements that are only in `A` but not in `B`.



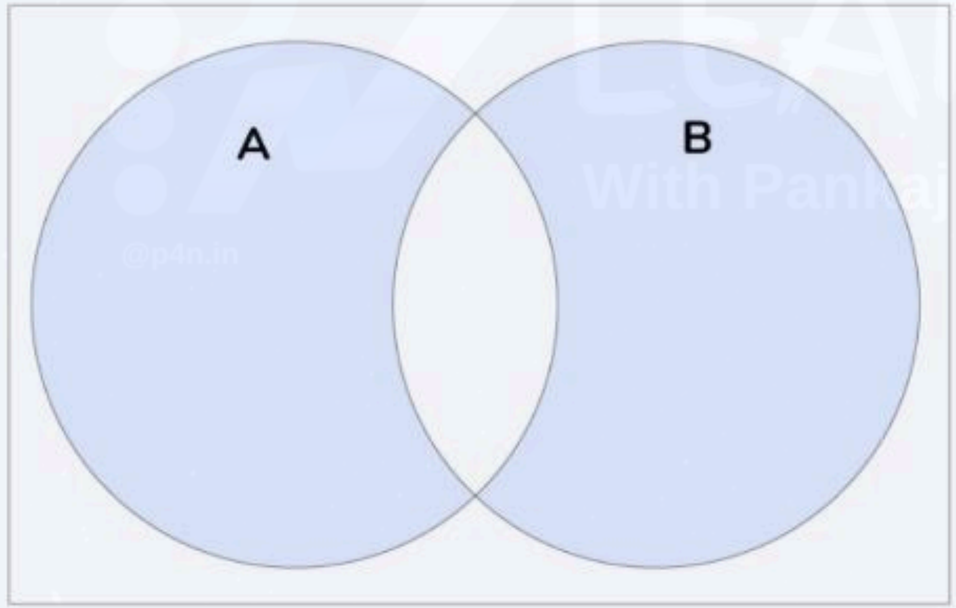
It is performed by `A - B`. Same can be accomplished using the `difference()` method.

```
A = {1, 2, 3}
B = {2, 3, 4, 5}

result = A - B
print(result)    # Output: {1}

result = A.difference(B)
print(result)    # Output: {1}

result = B - A
print(result)    # Output: {4, 5}
```

It's important to note that `A-B` may not be equal to `B-A`.

Set Symmetric Difference

The symmetric difference of two sets `A` and `B` is a set of elements in both `A` and `B` except those that are common in both.



It is performed by `^` operator. Same can be accomplished using the `symmetric_difference()` method.

```
A = {1, 2, 3}
B = {2, 3, 4, 5}

result = A ^ B
print(result)    # Output: {1, 4, 5}

result = A.symmetric_difference(B)
print(result)    # Output: {1, 4, 5}
```

Python Set Methods

Here's a list of set methods we have discussed till now.

- `add()` - to add a single element to a set
- `update()` - to add multiple elements to a set
- `remove()` - to remove an element from a set
- `discard()` - to remove an element from a set
- `pop()` - to remove and return an arbitrary element from a set
- `clear()` - to remove all elements from a set
- `union()` - returns union of two sets
- `intersection()` - returns intersection of two sets
- `difference()` - returns difference of two sets
- `symmetric_difference()` - returns symmetric difference of two sets

Iterating through a set

Using a `for` loop, we can iterate through each item in a set.

```
for letter in set('apple'):
    print(letter)
```

Output

```
l
a
p
e
```

Since elements of a set are unordered, you may get different output (having different order).

Check if an element exists

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

```
my_set = set("apple")

print('a' in my_set)  # Output: True
print('z' in my_set)  # Output: False

print('l' not in my_set)  # Output: False
```