# Sets

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

However, a set itself is mutable. We can add or remove items from it.

Since sets are unordered, we cannot access items using indexes like we do in lists.

Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together, or we can get the list of elements by looping through the set.

```
# Declaring and empty set
s = set()
print(type(s))

# Initializing a set
s = {1,2,3,4,1}
print(s)

# Set methods
# Adding a new element to the set
x = set()
x.add(1)
x.add(2)
x.add(1)
x.add(3)
print(x)


# By default set elemenates the duplicate values
s =
{1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5,5,6,6,6,6,6,7,7,7,7,7
}
print(s)
```

```
# filter the duplicate elements from a list, using set
list =
[1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5,5,6,6,6,6,6,7,7,7,7,7
]
print(list) #prints all the list along with duplicates

# First convert the list to a set using 'set()' function
for i in set(list):
    print(i)
```

```
# filter the duplicate elements from a list, using set
# Finding length of a set
s = {1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5,5,6,6,6,6,6,7,7,7,7,7}
print(s) #prints all the list along with duplicates
print(len(s))
```

**<u>Set Methods:</u>**

```
# Check if "banana" is present in the set:
s = {"apple", "banana", "cherry"}
print("banana" in s)


# To add items from another set into the current set, use the update()
method.
s = {"apple", "banana", "cherry"}
seasonal = {"pineapple", "mango", "papaya"}
s.update(seasonal)
print(s)

# The object in the update() method does not have to be a set, it can
be any iterable object (tuples, lists, dictionaries etc.).
s = {"apple", "banana", "cherry"}
l = ["kiwi", "orange"]
s.update(l)
print(s)

# Use the union() method that returns a new set containing all items
from both sets, or the update() method that inserts all the items from
one set into another:
set1 = {"a", "b" , "c"}
```

```python
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)


# The intersection_update() method will keep only the items that are
present in both sets.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)
print(x)


# The intersection() method will return a new set, that only contains
the items that are present in both sets.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.intersection(y)
print(z)


# Return a set that contains the items that only exist in set x, and
not in set y:
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.difference(y)
print(z)


# The symmetric_difference_update() method will keep only the elements
that are NOT present in both sets.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.symmetric_difference_update(y)
print(x)
```

```
# Remove the items that exist in both sets and returns the first set
# The difference_update() method is different from the difference()
method, because the difference() method returns a new set, without the
unwanted items, and the difference_update() method removes the
unwanted items from the original set.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.difference_update(y)
print(x)


# The symmetric_difference() method will return a new set, that
contains only the elements that are NOT present in both sets.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)
print(z)


# To remove an item in a set, use the remove(), or the discard()
method.
# Note: If the item to remove does not exist, remove() will raise an
error.
# You can also use the pop() method to remove an item, but this method
will remove the last item. Remember that sets are unordered, so you
will not know what item that gets removed.
# The return value of the pop() method is the removed item.
s = {"apple", "banana", "cherry"}
s.remove("banana")
print(s)
s.discard("cherry")
print(s)


# The clear() method empties the set:
# The del keyword will delete the set completely:
s = {"apple", "banana", "cherry"}
s.clear()
print(s)



# More Methods on Sets
fruits = {"apple", "banana", "cherry"}
```

```python
x = fruits.copy()
print(x)

# Returns True if no items in set x is present in set y:
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "facebook"}

z = x.isdisjoint(y)
print(z)

# Returns True if all items in set x are present in set y:
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)
print(z)


# Returns True if all items set y are present in set x:
x = {"f", "e", "d", "c", "b", "a"}
y = {"a", "b", "c"}

z = x.issuperset(y)
print(z)
```

.................................................................................

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

# set cannot have duplicates
# Output: {1, 2, 3, 4}
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)

# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1, 2, 3, 2])
print(my_set)

# set cannot have mutable items
# here [3, 4] is a mutable list
# this will cause an error.
```

```
my_set = {1, 2, [3, 4]}


# initialize my_set
my_set = {1, 3}
print(my_set)

# my_set[0]
# if you uncomment the above line
# you will get an error
# TypeError: 'set' object does not support indexing

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

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2, 3, 4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4, 5], {1, 6, 8})
print(my_set)
```

---

```
# Built-in Functions with Set
# The all() function returns True if all elements in the given
iterable are true. If not, it returns False.

boolean_list = ['True', 'True', 'True']
result = all(boolean_list)
print(result)


# The any() function returns True if any element of an iterable is
True. If not, it returns False.
boolean_list = ['True', 'False', 'True']
result = any(boolean_list)
print(result)
```

```
# The enumerate() method adds a counter to an iterable and returns it
(the enumerate object).
languages = ['Python', 'Java', 'JavaScript']
enumerate_prime = enumerate(languages)
print(list(enumerate_prime))

enumerate_prime = enumerate(languages,11)
print(list(enumerate_prime))


# Looping Over an Enumerate object
grocery = ['bread', 'milk', 'butter']

for item in enumerate(grocery):
  print(item)

print('\n')

for count, item in enumerate(grocery):
  print(count, item)

print('\n')
# changing default start value
for count, item in enumerate(grocery, 100):
  print(count, item)
```

**Python frozenset():**

```
# Frozen set is just an immutable version of a Python set object. While elements
of a set can be modified at any time, elements of the frozen set remain the same
after creation.
# tuple of vowels
vowels = ('a', 'e', 'i', 'o', 'u')

fSet = frozenset(vowels)
print('The frozen set is:', fSet)
print('The empty frozen set is:', frozenset())

# frozensets are immutable
fSet.add('v')
```

## Set Operations in Conventional Way:

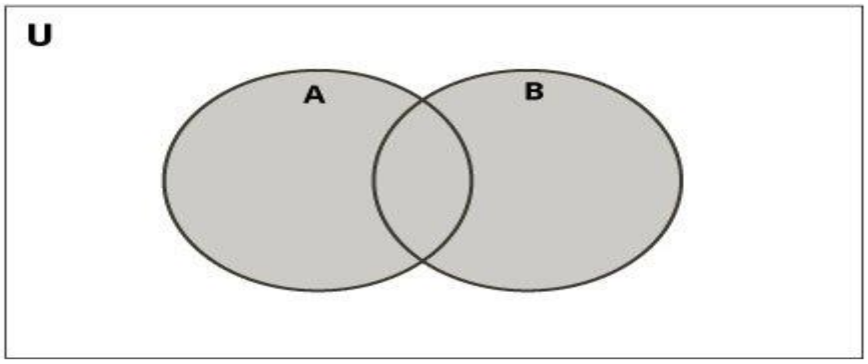
### Python Set Operations

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



Set Union in Python

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 `union()` method.

```
# Set union method
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

**Output**

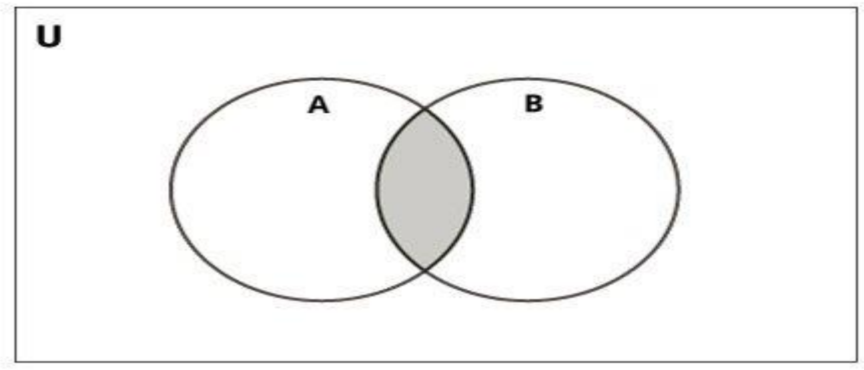
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Try the following examples on Python shell.

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

# use union function on B
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

## Set Intersection



Set Intersection in Python

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

```
# Intersection of sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)
```

**Output**

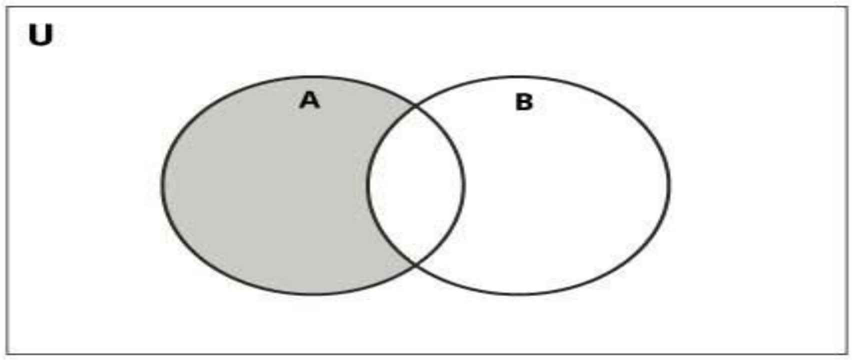
```
{4, 5}
```

Try the following examples on Python shell.

```
# use intersection function on A
>>> A.intersection(B)
{4, 5}

# use intersection function on B
>>> B.intersection(A)
{4, 5}
```

## Set Difference



Set Difference in Python

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

```
# Difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

**Output**

```
{1, 2, 3}
```

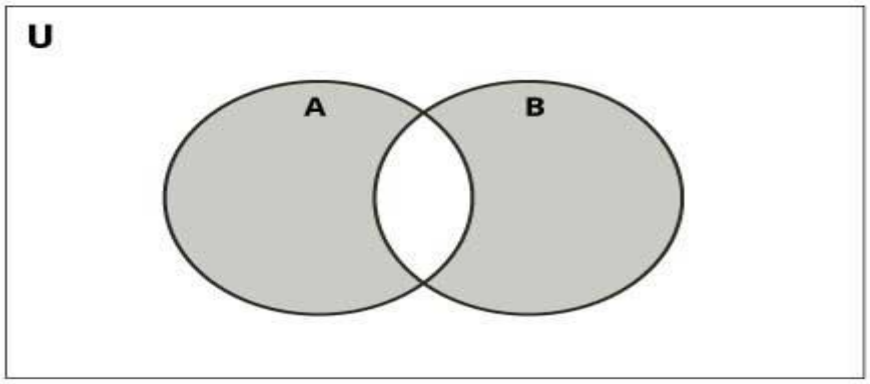
Try the following examples on Python shell.

```
# use difference function on A
>>> A.difference(B)
{1, 2, 3}

# use - operator on B
>>> B - A
{8, 6, 7}
```

```
# use difference function on B
>>> B.difference(A)
{8, 6, 7}
```

## Set Symmetric Difference



Set Symmetric Difference in Python

Symmetric Difference of `A` and `B` is a set of elements in `A` and `B` but not in both (excluding the intersection).
Symmetric difference is performed using `^` operator. Same can be accomplished using the method `symmetric_difference()`.

```
# Symmetric difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use ^ operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

### Output

```
{1, 2, 3, 6, 7, 8}
```

Try the following examples on Python shell.

```
# use symmetric_difference function on A
>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}

# use symmetric_difference function on B
>>> B.symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```