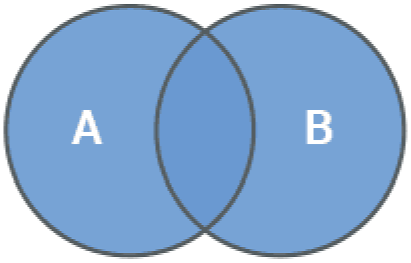
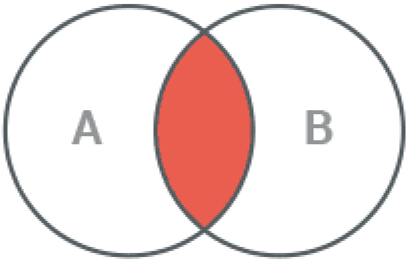
Python Set

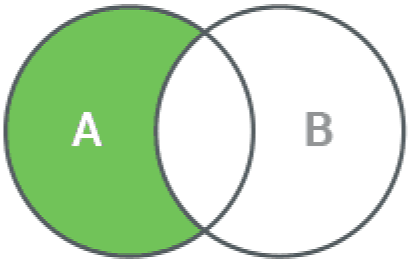
Python set is an unordered collection of unique items. They are commonly used for computing mathematical operations such as union, intersection, difference, and symmetric difference.



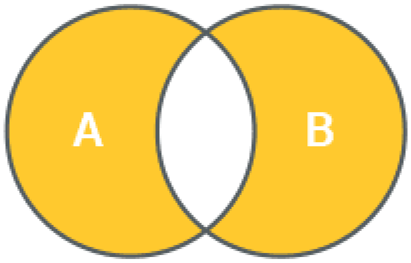
Union



Intersection



Difference



Symmetric Difference

The important properties of Python sets are as follows:

- **Sets are unordered – Items stored in a set aren't kept in any particular order.**
- **Set items are unique – Duplicate items are not allowed.**
- **Sets are unindexed – You cannot access set items by referring to an index.**
- **Sets are changeable (mutable) – They can be changed in place, can grow and shrink on demand.**

Create a Set

You can create a set by placing a comma-separated sequence of items in curly braces `{}`.

```
# A set of strings

S = {'red', 'green', 'blue'}


# A set of mixed datatypes
S = {1, 'abc', 1.23, (3+4j), True}
```

Sets don’t allow duplicates. They are automatically removed during the creation of a set.

```
S = {'red', 'green', 'blue', 'red'}

print(S)
# Prints {'blue', 'green', 'red'}
```

A set itself is mutable (changeable), but it cannot contain mutable objects. Therefore, immutable objects like numbers, strings, tuples can be a set item, but lists and dictionaries are mutable, so they cannot be.

```
S = {1, 'abc', ('a', 'b'), True}

S = {[1, 2], {'a':1, 'b':2}}
# Triggers TypeError: unhashable type: 'list'
```

Set constructor

You can also create a set using a type constructor called set().

```
# Set of items in an iterable

S = set('abc')

print(S)

# Prints {'a', 'b', 'c'}


# Set of successive integers

S = set(range(0, 4))

print(S)

# Prints {0, 1, 2, 3}
# Convert list into set
S = set([1, 2, 3])
print(S)
# Prints {1, 2, 3}
```

Add Items to a Set

You can add a single item to a set using add() method.

```
S = {'red', 'green', 'blue'}
S.add('yellow')
print(S)
# Prints {'blue', 'green', 'yellow', 'red'}
```

You can add multiple items to a set using update() method.

```
S = {'red', 'green', 'blue'}
S.update(['yellow', 'orange'])
print(S)
# Prints {'blue', 'orange', 'green', 'yellow', 'red'}
```

Remove Items from a Set

To remove a single item from a set, use remove() or discard() method.

```
# with remove() method
S = {'red', 'green', 'blue'}
S.remove('red')
print(S)
# Prints {'blue', 'green'}

# with discard() method
S = {'red', 'green', 'blue'}
S.discard('red')
```

```
print(S)
# Prints {'blue', 'green'}
```

remove() vs discard()

Both methods work exactly the same. The only difference is that If specified item is not present in a set:

- **remove() method raises** KeyError
- **discard() method does nothing**

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

```
S = {'red', 'green', 'blue'}
x = S.pop()
print(S)
# Prints {'green', 'red'}

# removed item
print(x)
# Prints blue
```

Use clear() method to remove all items from the set.

```
S = {'red', 'green', 'blue'}
S.clear()
print(S)
# Prints set()
```

Find Set Size

To find how many items a set has, use len() method.

```
S = {'red', 'green', 'blue'}
print(len(S))
# Prints 3
```

Iterate Through a Set

To iterate over the items of a set, use a simple for loop.

```
S = {'red', 'green', 'blue'}
for item in S:
    print(item)
# Prints blue green red
```

Check if Item Exists in a Set

To check if a specific item is present in a set, you can use in and not in operators with if statement.

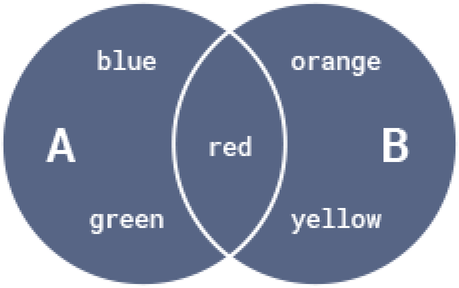
```
# Check for presence
S = {'red', 'green', 'blue'}
if 'red' in S:
    print('yes')


# Check for absence
S = {'red', 'green', 'blue'}
if 'yellow' not in S:
    print('yes')
```

Set Operations

Sets are commonly used for computing mathematical operations such as intersection, union, difference, and symmetric difference.

Set Union

You can perform union on two or more sets using union() method or `|` operator.

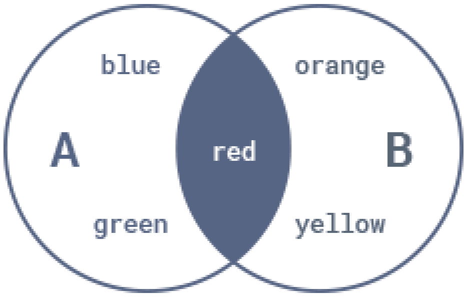


Union of the sets A and B is the set of all items in either A or B

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

# by operator
print(A | B)
# Prints {'blue', 'green', 'yellow', 'orange', 'red'}

# by method
print(A.union(B))
# Prints {'blue', 'green', 'yellow', 'orange', 'red'}
```

Set Intersection

You can perform intersection on two or more sets using intersection() method or `&` operator.



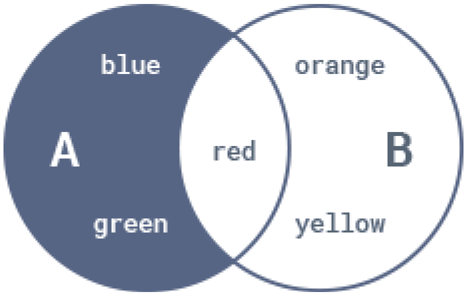
Intersection of the sets A and B is the set of items common to both A and B.

```
A = {'red', 'green', 'blue'}
```

```
B = {'yellow', 'red', 'orange'}


# by operator
print(A & B)
# Prints {'red'}


# by method
print(A.intersection(B))
# Prints {'red'}
```

Set Difference

You can compute the difference between two or more sets using difference() method or `-` operator.

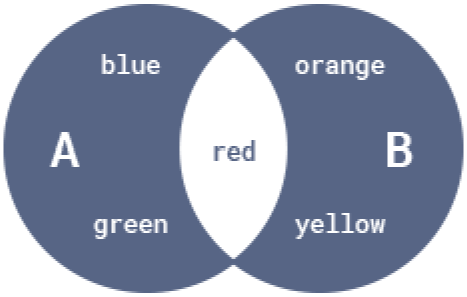


Set Difference of A and B is the set of all items that are in A but not in B.

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}


# by operator
print(A - B)
# Prints {'blue', 'green'}


# by method
print(A.difference(B))
# Prints {'blue', 'green'}
```

Set Symmetric Difference

You can compute symmetric difference between two or more sets using symmetric_difference() method or `^` operator.



Symmetric difference of sets A and B is the set of all elements in either A or B, but not both.

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

# by operator
print(A ^ B)
# Prints {'orange', 'blue', 'green', 'yellow'}

# by method
print(A.symmetric_difference(B))
# Prints {'orange', 'blue', 'green', 'yellow'}
```

Other Set Operations

Below is a list of all set operations available in Python.

Method	Description
union()	**Return a new set containing the union of two or more sets**
update()	**Modify this set with the union of this set and other sets**
intersection()	**Returns a new set which is the intersection of two or more sets**
intersection_update()	**Removes the items from this set that are not present in other sets**
difference()	**Returns a new set containing the difference between two or more sets**
difference_update()	**Removes the items from this set that are also included in another set**
symmetric_difference()	**Returns a new set with the symmetric differences of two or more sets**

symmetric_difference_update() **Modify this set with the symmetric difference of this set and other set**
isdisjoint() **Determines whether or not two sets have any elements in common**
issubset() **Determines whether one set is a subset of the other**
issuperset() **Determines whether one set is a superset of the other**

Python Frozenset

Python provides another built-in type called a frozenset. Frozenset is just like set, only immutable (unchangeable).

You can create a frozenset using frozenset() method. It freezes the given sequence and makes it unchangeable.

```
S = frozenset({'red', 'green', 'blue'})

print(S)
# Prints frozenset({'green', 'red', 'blue'})
```

As frozensets are unchangeable, you can perform non-modifying operations on them.

```
# finding size

S = frozenset({'red', 'green', 'blue'})

print(len(S))

# Prints 3


# performing union

S = frozenset({'red', 'green', 'blue'})

print(S | {'yellow'})
# Prints frozenset({'blue', 'green', 'yellow', 'red'})
```

However, methods that attempt to modify a frozenset will raise error.

```
# removing an item

S = frozenset({'red', 'green', 'blue'})

S.pop()

# Triggers AttributeError: 'frozenset' object has no attribute 'pop'
# adding an item

S = frozenset({'red', 'green', 'blue'})

S.add('yellow')
# Triggers AttributeError: 'frozenset' object has no attribute 'add'
```

Unlike sets, frozensets are unchangeable so they can be used as keys to a dictionary.

For example, D = {frozenset(['dev','mgr']):'Bob'}

Built-in Functions with Set

Below is a list of all built-in functions that you can use with set objects.

Method	Description
all()	Returns True if all items in a set are true
any()	Returns True if any item in a set is true
enumerate()	Takes a set and returns an enumerate object
len()	Returns the number of items in the set
max()	Returns the largest item of the set
min()	Returns the smallest item of the set
sorted()	Returns a sorted set
sum()	Sums items of the set