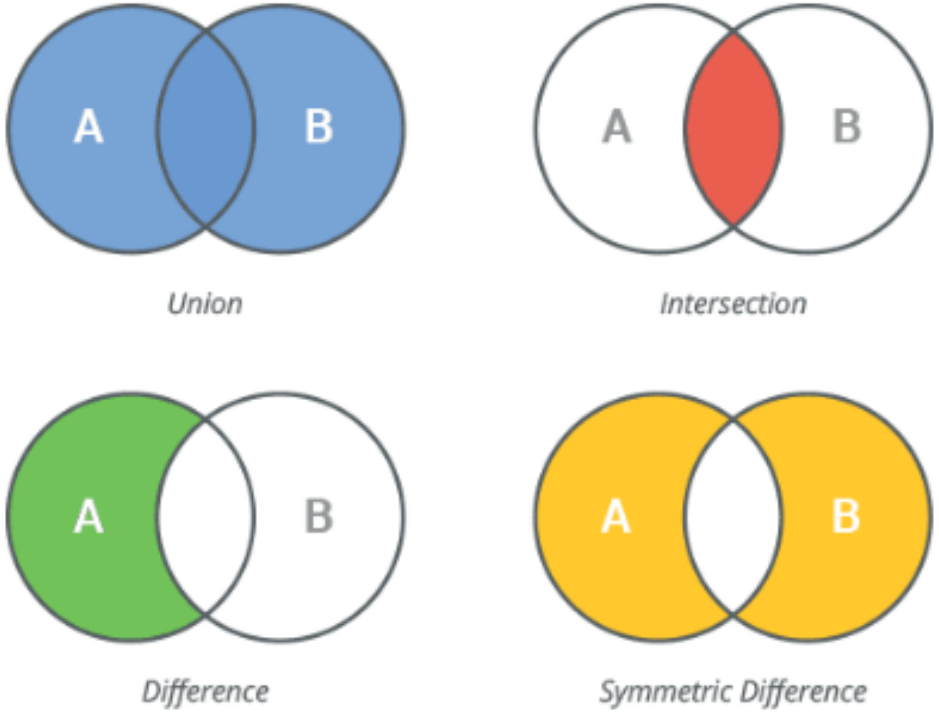
Set in Python

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.



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'
```

Using Set () to create 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.

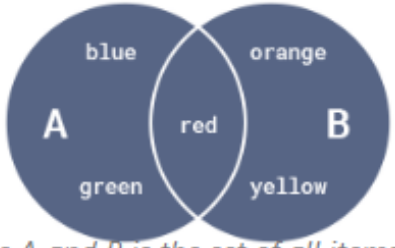
```
# Check for presence
S = {'red', 'green', 'blue'}
if 'red' in S:
    print('yes')

# 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.

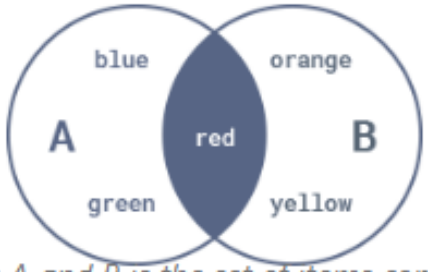


Union of the sets A and B is the set of all items in either A or B

```
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.

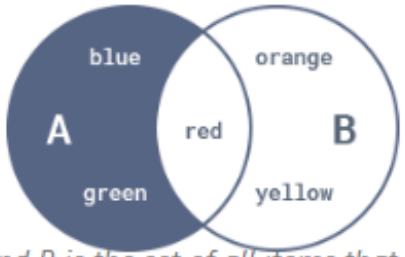


Intersection of the sets A and B is the set of items common to both A and B.

```
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'}
```

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 Set Methods

Python Set intersection_update() Method

Usage

The `intersection_update()` method updates the set by removing the items that are not common to all the specified sets.

You can specify as many `sets` as you want, just separate each set with a comma.

If you don't want to update the original set, use intersection() method.

Syntax

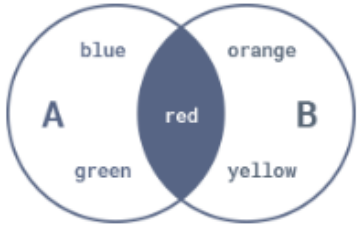
```
set.intersection_update(set1,set2…)
```

Parameter	Condition	Description
`set1, set2…`	Optional	A comma-separated list of one or more sets to search for common items in

Python set intersection_update() method parameters

Basic Example

```
# Remove items that are not common to both A & B
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

A.intersection_update(B)
print(A)
# Prints {'red'}
```



Equivalent Operator &=

You can achieve the same result by using the `&=` augmented assignment operator.

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

A &= B
print(A)
# Prints {'red'}
```

intersection_update() Method with Multiple Sets

Multiple sets can be specified with either the operator or the method.

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

# by method
A.intersection_update(B,C)
print(A)
# Prints {'red'}

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

Python Set difference_update() Method

Usage

The `difference_update()` method updates the set by removing items found in specified sets.

You can specify as many `sets` as you want, just separate each set with a comma.

If you don't want to update the original set, use difference() method.

Syntax

```
set.difference_update(set1,set2…)
```

Parameter	Condition	Description
`set1, set2…`	Optional	A comma-separated list of one or more sets to find differences in

Python set difference_update() method parameters

Syntax

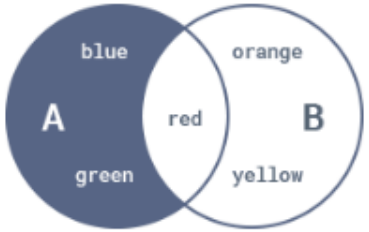
```
set.difference_update(set1,set2…)
```

Parameter	Condition	Description
set1, set2…	Optional	A comma-separated list of one or more sets to find differences in

Python set difference_update() method parameters

Basic Example

```
# Remove items from A found in B
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

A.difference_update(B)
print(A)
# Prints {'blue', 'green'}
```



Equivalent Operator -=

You can achieve the same result by using the `-=` augmented assignment operator.

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

A -= B
print(A)
# Prints {'blue', 'green'}
```

difference_update() Method with Multiple Sets

Multiple sets can be specified with either the operator or the method.

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

# by method
A.difference_update(B,C)
print(A)
# Prints {'green'}

# by operator
A -= B | C
print(A)
# Prints {'green'}
```

Python Set symmetric_difference_update()

Usage

The `symmetric_difference_update()` method updates the set by keeping only elements found in either set, but not in both.

If you don't want to update the original set, use symmetric_difference() method.

> The symmetric difference is actually the union of the two sets, minus their intersection.

Syntax

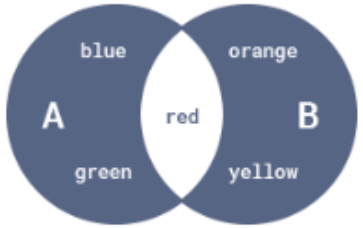
```
set.symmetric_difference_update(set)
```

Parameter	Condition	Description
set	Required	A set to find difference in

Python set symmetric_difference_update() method parameters

Basic Example

```
# Update A by adding items from B, except common items
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

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



Equivalent Operator ^=

You can achieve the same result by using the `^=` augmented assignment operator.

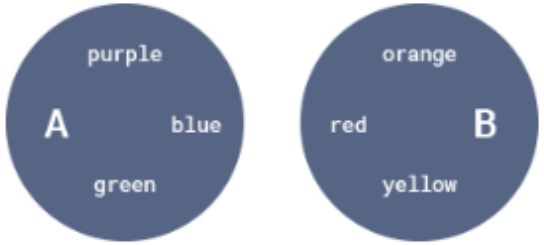
```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

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

Python Set isdisjoint() Method

Usage

The `isdisjoint()` method returns True if two sets have no items in common, otherwise FALSE.

> Sets are disjoint if and only if their intersection is the empty set.



Syntax

```
set.isdisjoint(set)
```

Parameter	Condition	Description
set	Required	A set to search for common items in

Python set isdisjoint() method parameters

The method returns FALSE if the specified sets have any item in common.

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

print(A.isdisjoint(B))
# Prints False
```

Examples

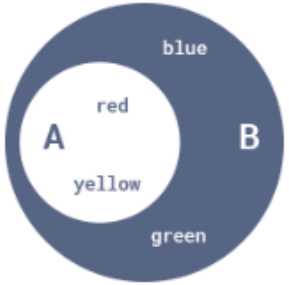
```
# Check if two sets have no items in common
A = {'green', 'blue', 'purple'}
B = {'yellow', 'red', 'orange'}

print(A.isdisjoint(B))
# Prints True
```

Python Set issubset() Method

Usage

The `issubset()` method returns True if all items in the set are present in the specified `set`, otherwise FALSE.

> In set theory, every set is a subset of itself.
>
> For example, `A.issubset(A)` is True.



Syntax

```
set.issubset(set)
```

Parameter	Condition	Description
set	Required	A set to search for common items in

Python set issubset() method parameters

Basic Example

```
# Check if all items in A are present in B
A = {'yellow', 'red'}
B = {'red', 'green', 'blue', 'yellow'}
print(A.issubset(B))
# Prints True
```

Equivalent Operator <=

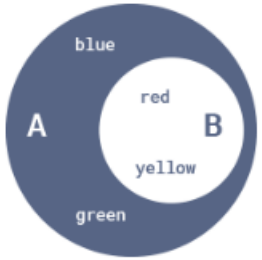
You can achieve the same result by using the `<=` comparison operator.

```
A = {'yellow', 'red'}
B = {'red', 'green', 'blue', 'yellow'}
print(A <= B)
# Prints True
```

Python Set issuperset() Method

Usage

The `issuperset()` method returns True if all items in the specified `set` are present in the original set, otherwise FALSE.



Syntax

```
set.issuperset(set)
```

Parameter	Condition	Description
set	Required	A set to search for common items in

Python set issuperset() method parameters

Equivalent Operator >=

You can achieve the same result by using the `>=` comparison operator.

```
A = {'red', 'green', 'blue', 'yellow'}
B = {'yellow', 'red'}
print(A >= B)
# Prints True
```

Basic Example

```
# Check if all items in B are present in A
A = {'red', 'green', 'blue', 'yellow'}
B = {'yellow', 'red'}
print(A.issuperset(B))
# Prints True
```