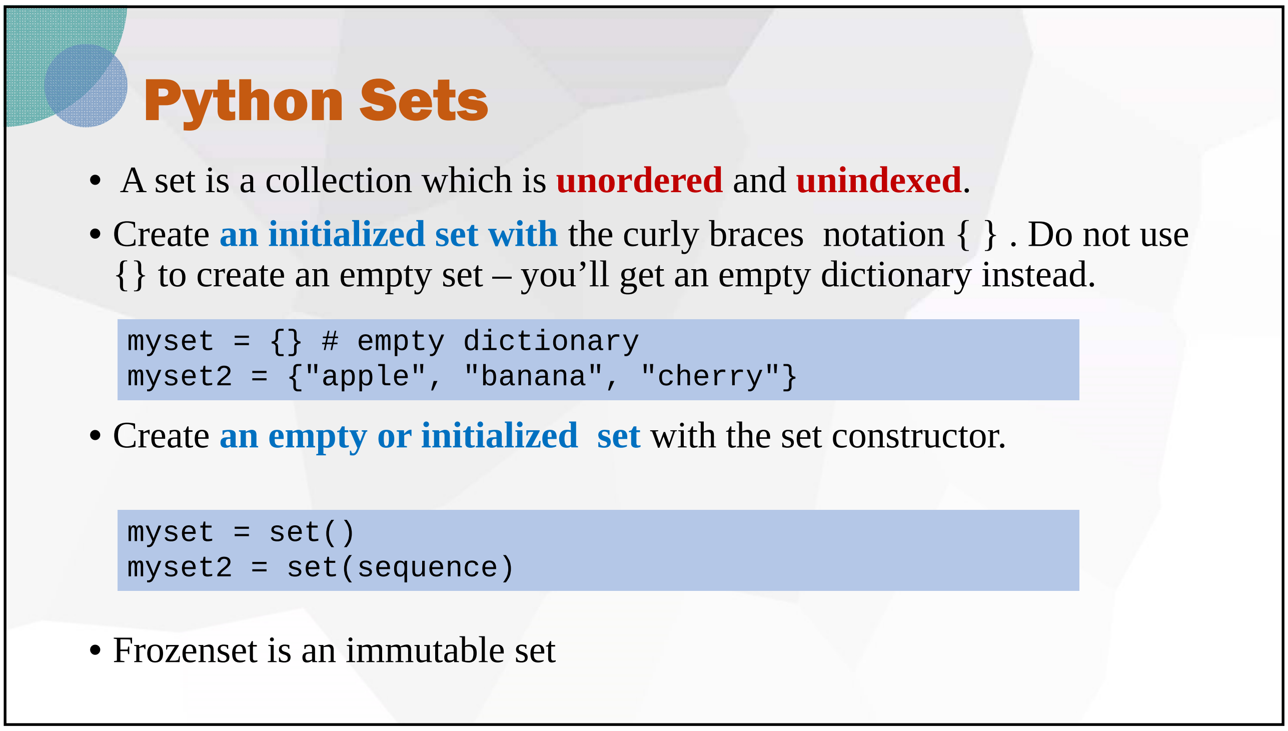


Python Sets

- A set is a collection which is **unordered** and **unindexed**.
- Create **an initialized set with** the curly braces notation { } . Do not use {} to create an empty set – you'll get an empty dictionary instead.

```
myset = {} # empty dictionary
myset2 = {"apple", "banana", "cherry"}
```

- Create **an empty or initialized set** with the set constructor.

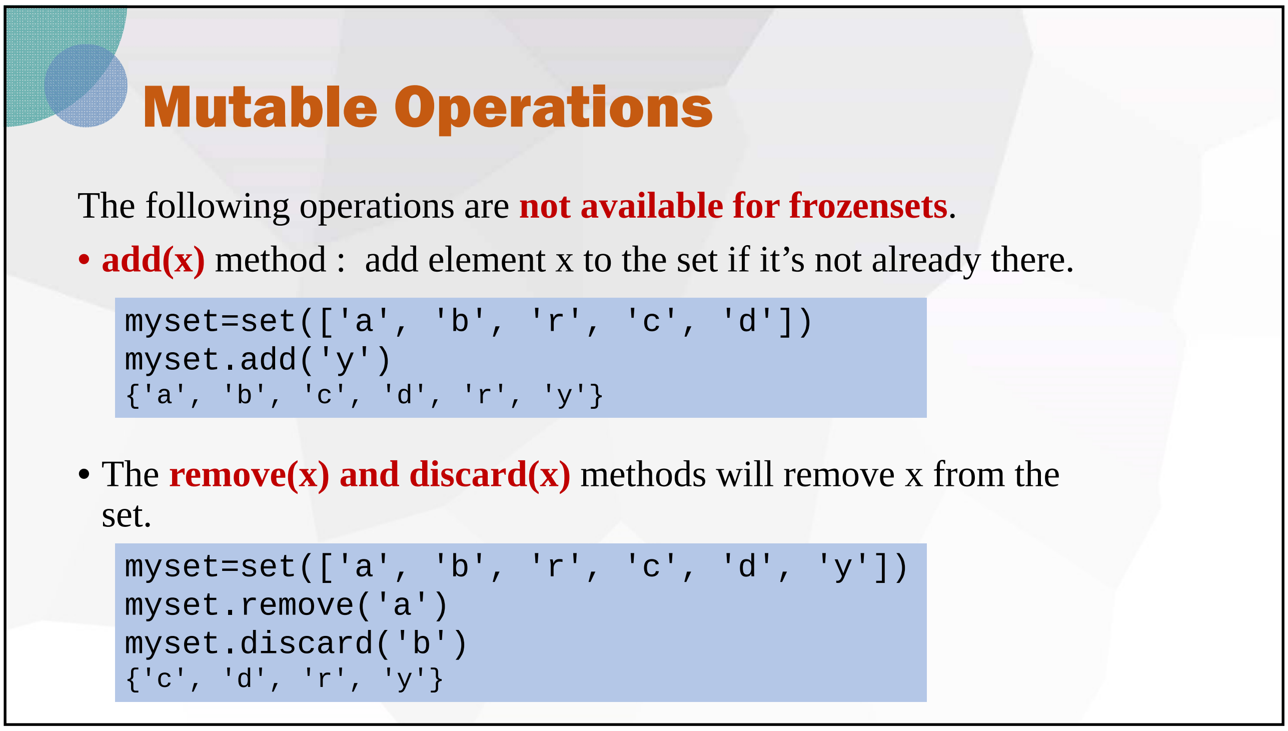
```
myset = set()
myset2 = set(sequence)
```

- Frozenset is an immutable set

Python Sets

- A frozenset is an immutable collection .
- Create **an empty or initialized frozenset** with the frozenset constructor.

```
myset = frozenset()
myset2 = frozenset(['A', 'E', 'I', 'O', 'U'])
myset2
```



Mutable Operations

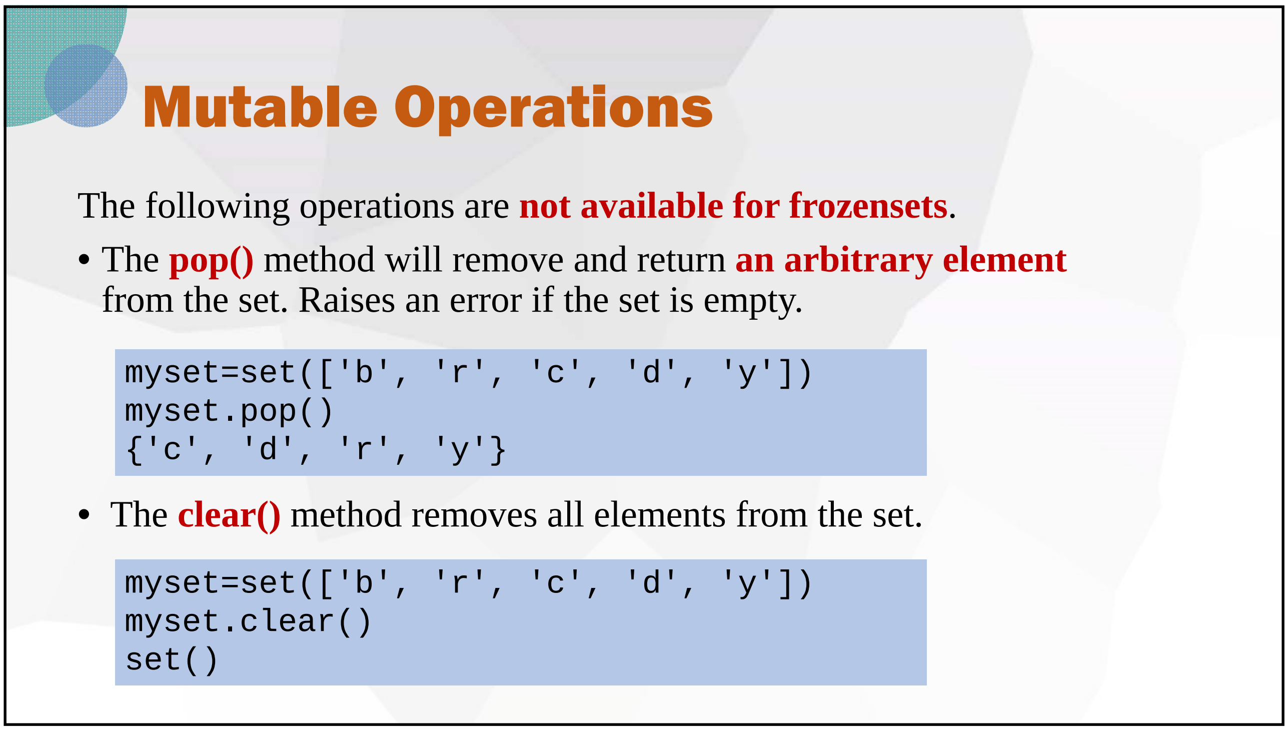
The following operations are **not available for frozensets**.

- **add(x)** method : add element x to the set if it's not already there.

```
myset=set(['a', 'b', 'r', 'c', 'd'])
myset.add('y')
{'a', 'b', 'c', 'd', 'r', 'y'}
```

- The **remove(x) and discard(x)** methods will remove x from the set.

```
myset=set(['a', 'b', 'r', 'c', 'd', 'y'])
myset.remove('a')
myset.discard('b')
{'c', 'd', 'r', 'y'}
```



Mutable Operations

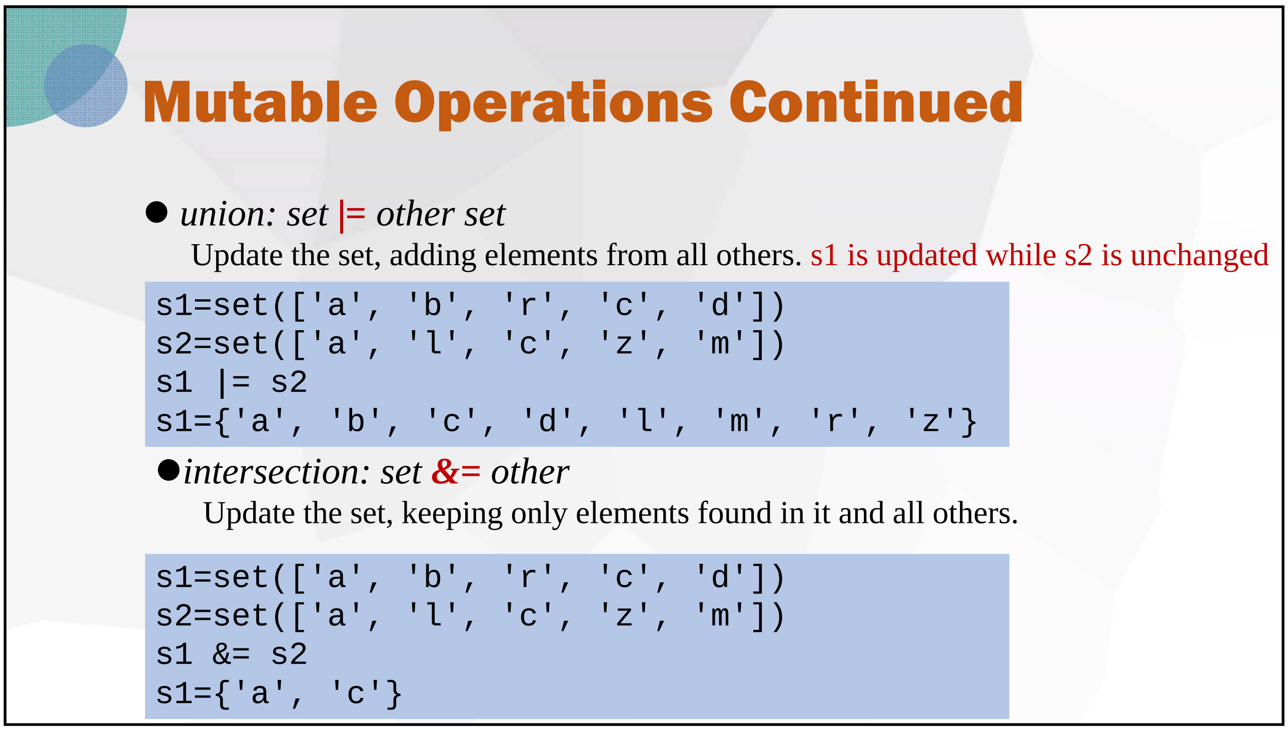
The following operations are **not available for frozensets**.

- The **pop()** method will remove and return **an arbitrary element** from the set. Raises an error if the set is empty.

```
myset=set(['b', 'r', 'c', 'd', 'y'])
myset.pop()
{'c', 'd', 'r', 'y'}
```

- The **clear()** method removes all elements from the set.

```
myset=set(['b', 'r', 'c', 'd', 'y'])
myset.clear()
set()
```



Mutable Operations Continued

- *union: set |= other set*

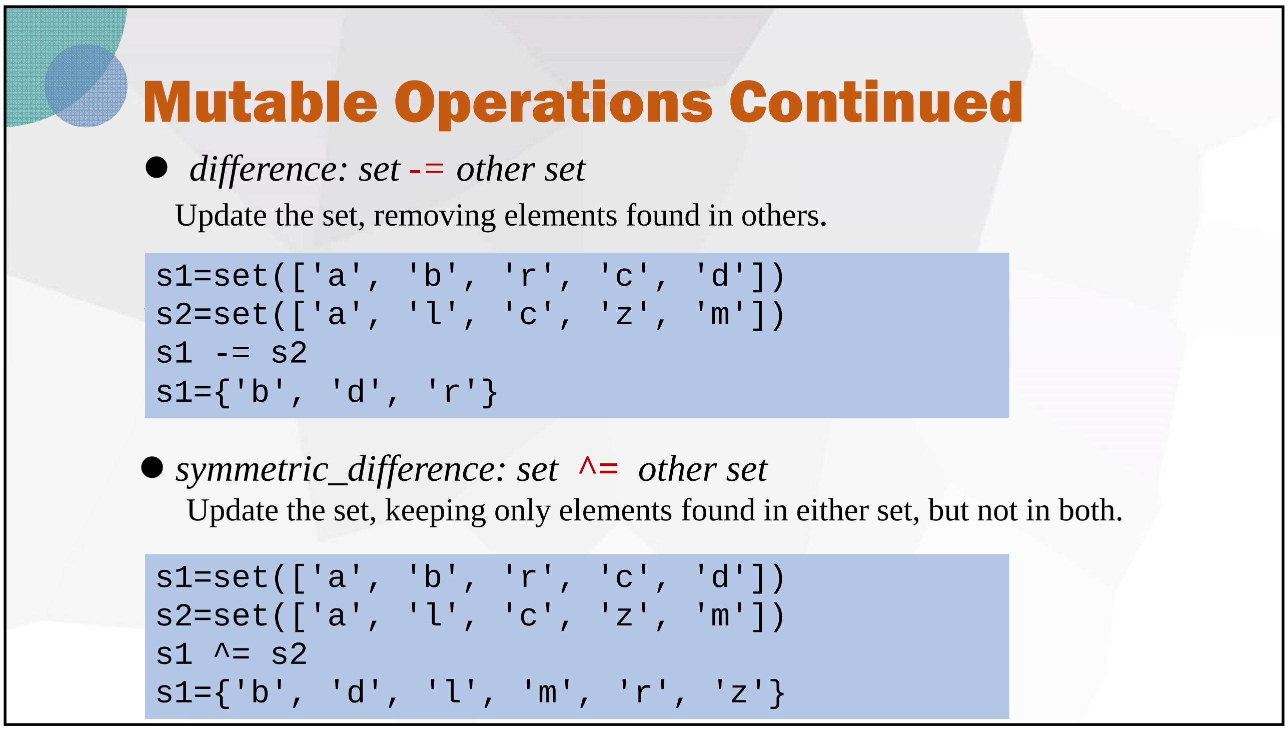
  Update the set, adding elements from all others. s1 is updated while s2 is unchanged

```
s1=set(['a', 'b', 'r', 'c', 'd'])
s2=set(['a', 'l', 'c', 'z', 'm'])
s1 |= s2
s1={'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}
```

- *intersection: set* **&=** *other*

  Update the set, keeping only elements found in it and all others.

```
s1=set(['a', 'b', 'r', 'c', 'd'])
s2=set(['a', 'l', 'c', 'z', 'm'])
s1 &= s2
s1={'a', 'c'}
```



Mutable Operations Continued

- *difference: set* -= *other set*

  Update the set, removing elements found in others.

```
s1=set(['a', 'b', 'r', 'c', 'd'])
s2=set(['a', 'l', 'c', 'z', 'm'])
s1 -= s2
s1={'b', 'd', 'r'}
```

- *symmetric_difference: set* ^= *other set*

  Update the set, keeping only elements found in either set, but not in both.

```
s1=set(['a', 'b', 'r', 'c', 'd'])
s2=set(['a', 'l', 'c', 'z', 'm'])
s1 ^= s2
s1={'b', 'd', 'l', 'm', 'r', 'z'}
```

Set Operations

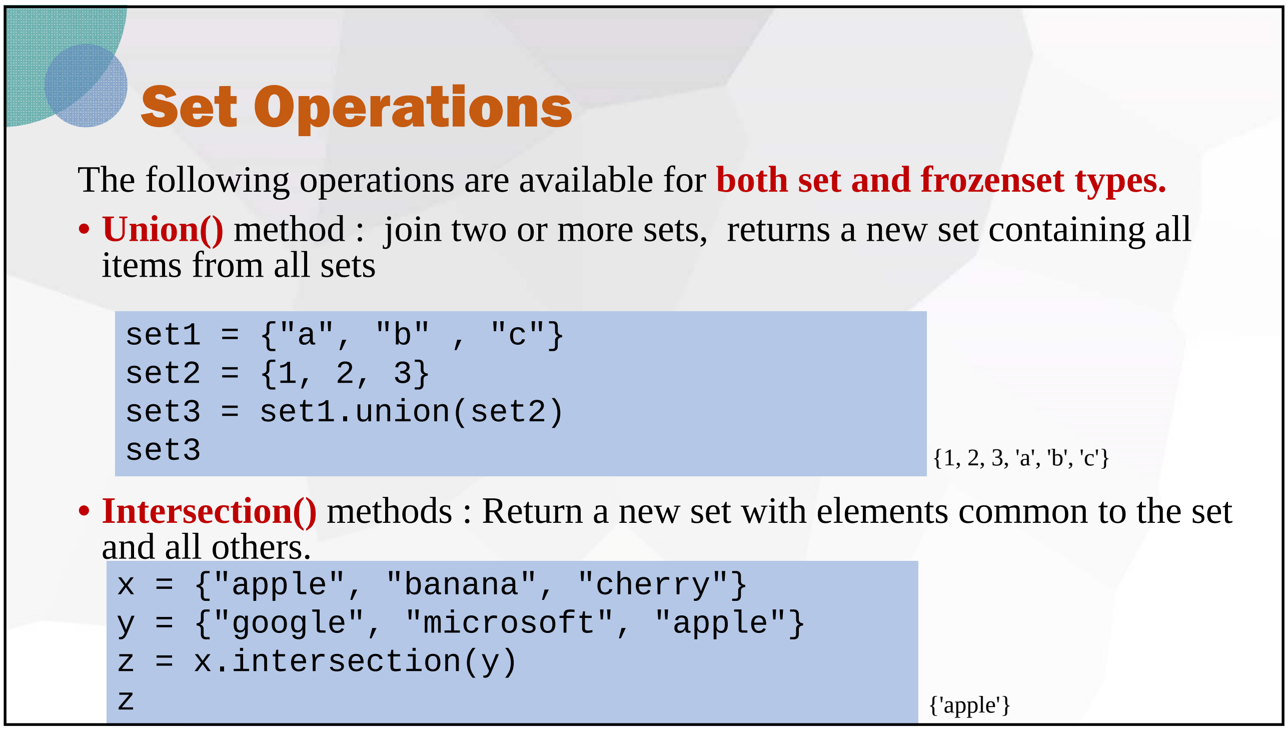
The following operations are available for **both set and frozenset types.**

- Comparison operators: **>=, <=**
  - ➢ test whether a set is a superset or subset of some other set.

●The > and < operators check for proper supersets/subsets.

```
s1 = set('abracadabra')
s2 = set('bard')
s1 >= s2
True
s1 > s2
True
s1 <= s2
False
```

```
s1 = set('abcd')
s2 = set('abcd')
s1 >= s2
True
s1 > s2
False
```



Set Operations

The following operations are available for **both set and frozenset types.**

- **Union()** method : join two or more sets, returns a new set containing all items from all sets

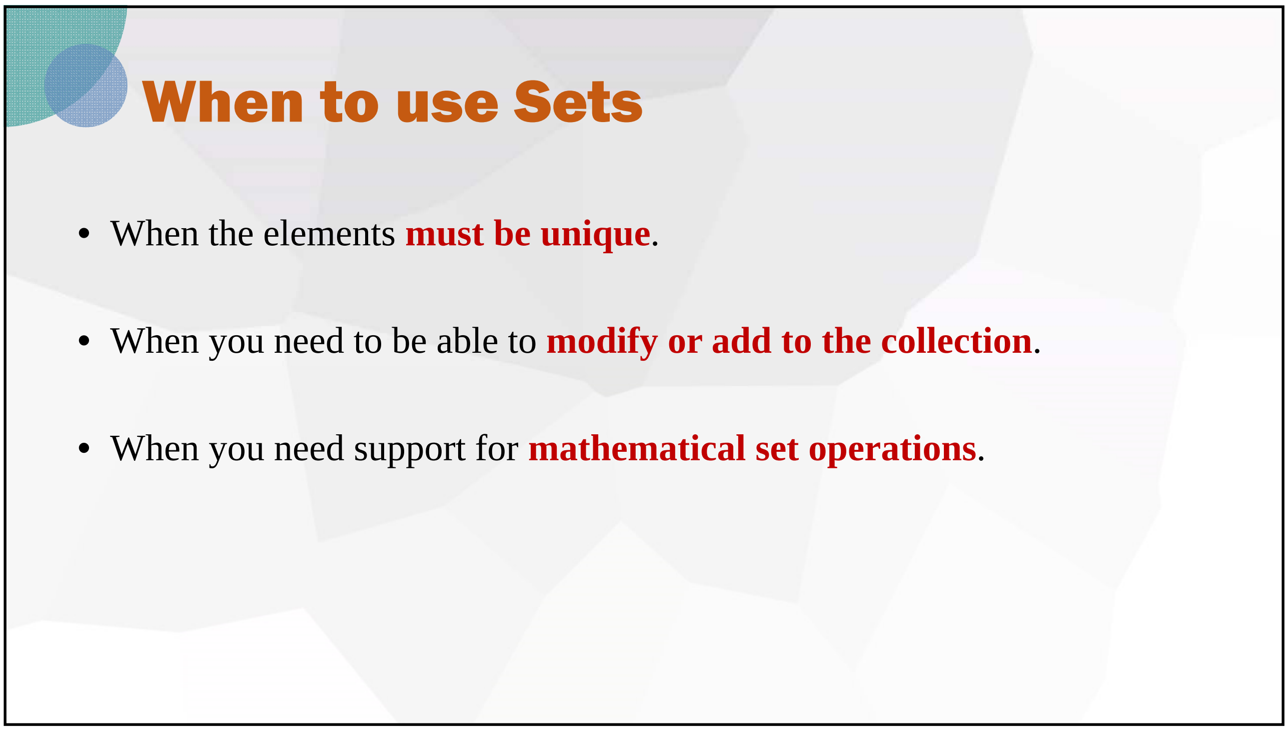
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
set3
```

```
{1, 2, 3, 'a', 'b', 'c'}
```

- **Intersection()** methods : Return a new set with elements common to the set and all others.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y)
z
```

```
{'apple'}
```



When to use Sets

- When the elements **must be unique**.
- When you need to be able to **modify or add to the collection**.
- When you need support for **mathematical set operations**.