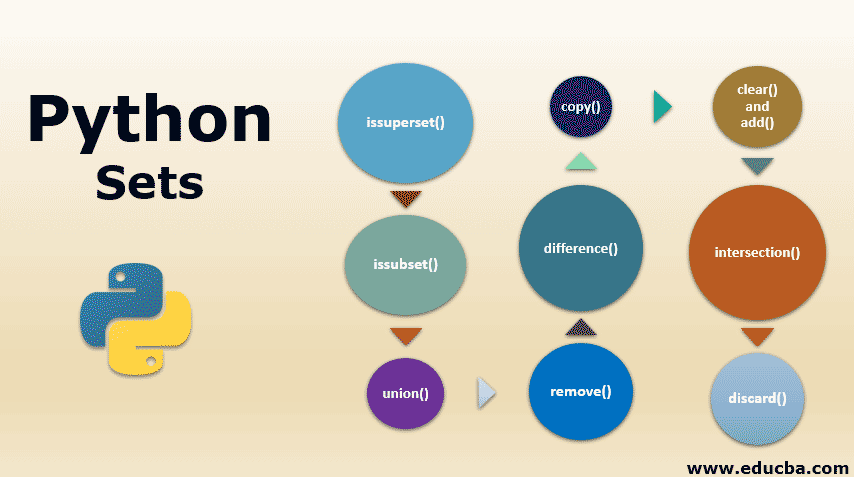
**Set Data Type:**

In Python, a set is an unordered collection of unique elements. Sets are mutable, meaning that you can add or remove elements from them. Sets are enclosed in curly braces `{}`, similar to dictionaries, but they contain only elements without key-value pairs.



**Example:**

```python

# Creating a set

my\_set = {1, 2, 3, 4, 5}

```

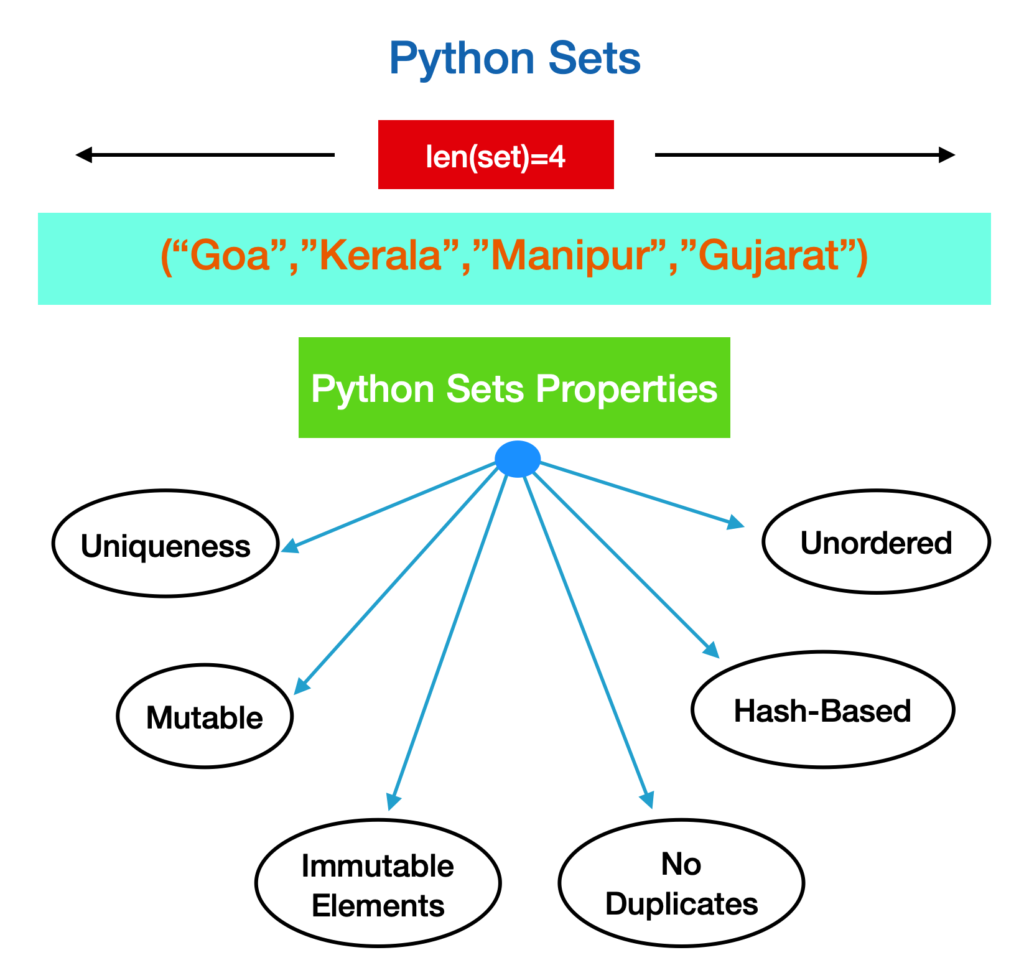
**Key Characteristics:**

1.**Mutable**: Sets can be modified after creation. You can add or remove elements from a set.

2.**Unordered**: Sets are unordered collections, meaning that the order of elements is not guaranteed.

3. **Unique Elements**: Sets contain only unique elements. Duplicate elements are automatically removed.

4. **Immutable Elements**: Elements in a set must be immutable objects, such as numbers, strings, or tuples.



**Common Operations and Methods:**

**1. Adding Elements:**

You can add elements to a set using the `add()` method or the `update()` method to add multiple elements.

```python

# Adding a single element

my\_set.add(6)

# Adding multiple elements

my\_set.update({7, 8, 9})

```

**2. Removing Elements:**

To remove elements from a set, you can use the `remove()` method, the `discard()` method (which won't raise an error if the element is not present), or the `pop()` method to remove and return an arbitrary element.

```python

# Removing a specific element

my\_set.remove(6)

# Removing an element safely

my\_set.discard(7)

# Removing and returning an arbitrary element

element = my\_set.pop()

```

**3. Set Methods:**

- **union()**: Returns a new set containing all unique elements from both sets.

- **intersection()**: Returns a new set containing common elements between two sets.

- **difference()**: Returns a new set containing elements that are present in the first set but not in the second set.

- **symmetric\_difference()**: Returns a new set containing elements that are present in either set, but not in both.

**4. Set Operations:**

1. **Union (`|`)**: Combines elements from two sets, excluding duplicates.
2. **Intersection (`&`)**: Returns elements common to both sets.
3. **Difference (`-`)**: Returns elements present in the first set but not in the second.
4. **Symmetric Difference (`^`)**: Returns elements present in either set, but not in both.

**# Define two sets**

set1 = {1, 2, 3, 4, 5}

set2 = {4, 5, 6, 7, 8}

**# Union (|)**

union\_set = set1 | set2

print("Union:", union\_set) # Output: {1, 2, 3, 4, 5, 6, 7, 8}

**# Intersection (&)**

intersection\_set = set1 & set2

print("Intersection:", intersection\_set) # Output: {4, 5}

**# Difference (-)**

difference\_set = set1 - set2

print("Difference (set1 - set2):", difference\_set) # Output: {1, 2, 3}

difference\_set = set2 - set1

print("Difference (set2 - set1):", difference\_set) # Output: {6, 7, 8}

**# Symmetric Difference (^)**

symmetric\_difference\_set = set1 ^ set2

print("Symmetric Difference:", symmetric\_difference\_set) # Output: {1, 2, 3, 6, 7, 8}

**Use Cases:**

**1. Removing Duplicates:** Sets are useful for removing duplicate elements from a list or collection of data.

2. **Membership Testing**: Check whether an element exists in a set efficiently using membership testing operations.

**3. Set Operations:** Perform set operations such as union, intersection, and difference to analyse data relationships.

**4. Filtering Data**: Use sets to filter out unwanted or redundant elements from datasets.

**5. Finding Unique Values**: Quickly find unique values in a dataset by converting it to a set.

Sets in Python provide a powerful tool for handling collections of unique elements and performing set operations efficiently. Understanding set methods and operations enables efficient manipulation and analysis of data in Python.