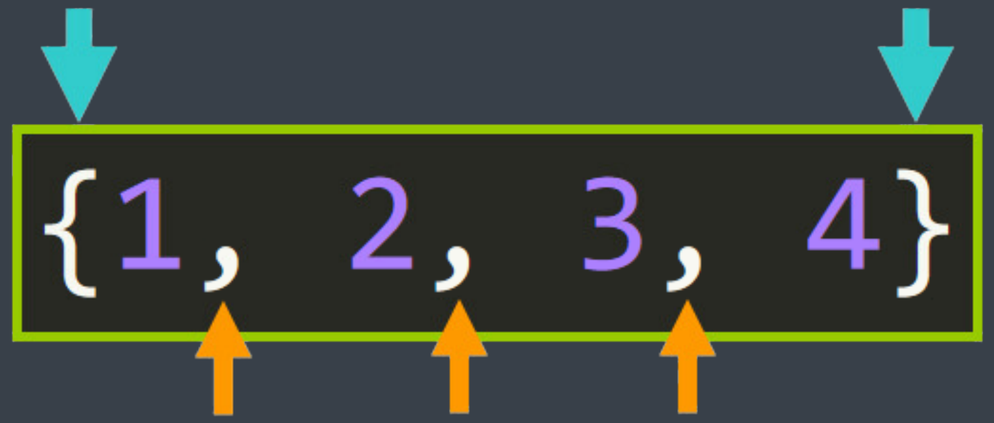
Sets

What is A Set ?



What is A Set ?

- Sets in Python are an unordered collection of unique elements
- Defined using curly braces {} or the set()
- Mutable
- Duplicate elements are automatically removed.

Creating Sets

Create a set by enclosing elements within curly braces {}. Alternatively, you can use the set()

```
# Creating an empty set
empty_set = set()

# Creating a set with elements
fruits = {'apple', 'banana', 'cherry'}

# Creating a set from a list
numbers = set([1, 2, 3, 4, 5])
```

Adding Elements

add() method to add a single element to a set

```
my_set = {1, 2}
my_set.add(3)
print(my_set)  # Output: {1, 2, 3}
```

Removing Elements

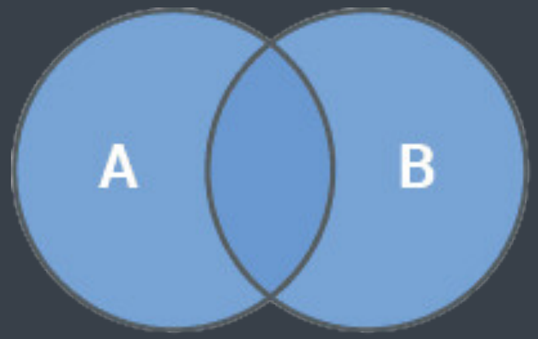
remove elements using the remove() or discard() methods.

```
my_set = {1, 2, 3}
my_set.remove(2)
print(my_set)  # Output: {1, 3}
```

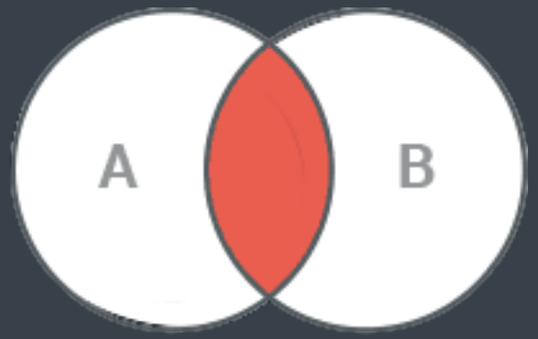
Set Operations

- Union
- Intersection
- Difference
- Symmetric difference

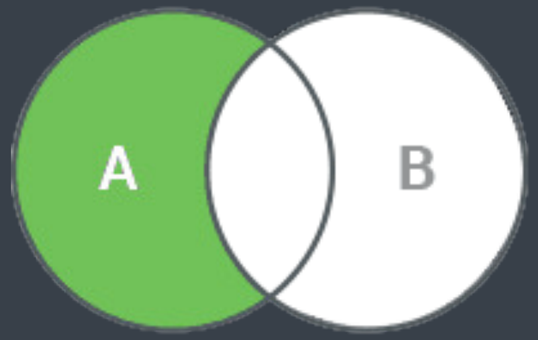
Set Operations



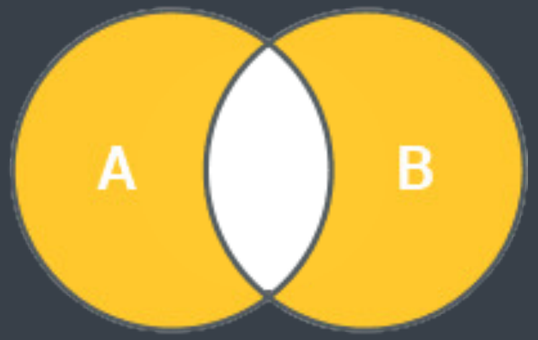
Union



Intersection



Difference



Symmetric difference

Set Operations

```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}

union_set = set1.union(set2)
print(union_set)  # Output: {1, 2, 3, 4, 5}

intersection_set = set1.intersection(set2)
print(intersection_set)  # Output: {3}

difference_set = set1.difference(set2)
print(difference_set)  # Output: {1, 2}

symmetric_difference_set = set1.symmetric_difference(set2)
print(symmetric_difference_set)  # Output: {1, 2, 4, 5}
```

Set Membership and Length

You can check if an element is present in a set using the **in** keyword. The **len()** function gives the number of elements in the set.

```
my_set = {1, 2, 3}
print(2 in my_set)  # Output: True
print(4 in my_set)  # Output: False

print(len(my_set))  # Output: 3
```

```
1 in {1, 2, 3, 4}
```

Element

Set

Frozen Sets

```
frozenset([1, 2, 3, 4])
```

Iterable

Frozen Sets

A frozenset is an immutable version of a set

```
my_set = {1, 2, 3}
frozen_set = frozenset(my_set)
```

Set Comprehensions

Like lists and dictionaries, sets also support comprehensions for concise set creation.

```
# Set comprehension to create a set of squares
squares = {x**2 for x in range(1, 6)}  # Output: {1, 4, 9, 16, 25}
```

Set Methods

Some common set methods include clear(), copy(), pop(), update(), etc.

```
my_set = {1, 2, 3}
my_set.clear()       # Removes all elements, resulting in an empty set
new_set = my_set.copy()  # Creates a shallow copy of the set
```

Dictionary

Defnition

key_1 : value_1

Defnition

A dictionary is an unordered collection of key-value pairs. Are used to store data in the form of key-value pairs, where each key is unique.

Creating Dictionaries

To create a dictionary, you use curly braces {} and specify key-value pairs separated by colons :. Keys and values can be of any data type.

```
# Creating a dictionary with name, age, and house information
my_dict = {'name': 'Harry', 'age': 11, 'house': 'Gryffindor'}

# Printing the dictionary
print(my_dict)
```

Creating Dictionaries

Keys	Values
name	"Harry"
age	11
house	"Gryffindor"

Accessing Values

You can access the values in a dictionary using square brackets [] with the key.

```
my_dict = {'name': 'Harry', 'age': 11, 'house': 'Gryffindor'}

# Accessing the values in the dictionary
name_value = my_dict['name']
age_value = my_dict['age']
house_value = my_dict['house']
```

Adding Values

To add new key-value pairs or update existing ones in a dictionary, you can use square brackets [] and the assignment operator =.

```
my_dict = {'name': 'Harry', 'age': 11, 'house': 'Gryffindor'}

# Adding a new key-value pair to the dictionary
my_dict['gender'] = 'Male'
```

Modifying Values

You can change the value associated with a key in a dictionary.

```
my_dict = {'name': 'Harry', 'age': 11, 'house': 'Gryffindor'}

# Modifying the 'age' value in the dictionary
my_dict['age'] = 12
```

Methods

Dictionaries have several useful methods,

- keys()
- values()
- items()
- get()
- pop()
- update()

Looping

You can use loops to iterate through the keys or values of a dictionary

```
for key in my_dict:
    print(key)

# Loop through values
for value in my_dict.values():
    print(value)

# Loop through key-value pairs
for key, value in my_dict.items():
    print(key, value)
```

Comprehensions

Similar to lists, dictionaries also support comprehensions for concise dictionary creation

```
squares_dict = {x: x**2 for x in range(1, 6)}
```

```
# Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Find the sum of all elements in a given list of numbers.

Sample Input: [10, 20, 30, 40, 50]

Sample Output: Sum of elements = 150

Find the maximum and minimum values in a list of numbers.

Sample Input: 15, 2, 7, 25, 10

Sample Output: Maximum = 25,
Minimum = 2

Count the number of occurrences of a specific element in a list.

Sample Input: [1, 2, 3, 2, 1, 4, 2, 5]
2

Sample Output: Count of 2 = 3

Given two sets, find their intersection (common elements) and union (all unique elements combined).

Sample Input:

Set A: {1, 2, 3, 4, 5}
Set B: {4, 5, 6, 7, 8}

Sample Output:

Intersection: {4, 5}
Union: {1, 2, 3, 4, 5, 6, 7, 8}

Create dictionaries, access values, update values, and iterate through key-value pairs.

Sample Input:

```
my_dict = { 'name': 'John', 'age': 30, 'city': 'New York' }
```

Sample Output:

```
name : John
age : 31
city : San Francisco
```