Introduction to Sets

A set is an unordered collection of *unique element*s in Python. Sets are mutable, meaning they can be modified after creation, but their elements must be immutable (like numbers, strings, or tuples).

Creating a Set

Syntax:

```
set_name = {element1, element2, element3, ...}  # Using curly braces

set_name = set(iterable)
```

Example:

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

   set2 = set([6, 7, 8, 9, 10])

   print("Set1:", set1)

   print("Set2:", set2)
```

Set Methods

add() **-** Adds a single element to the set. If the element is already present, the set remains unchanged.

Syntax:

```
        set_name.add(element)
```

Example:

```
   my_set = {1, 2, 3}

   my_set.add(4)

   print("After add:", my_set)
```

update() **-** Adds multiple elements from an iterable (like a list or tuple) to the set.

Syntax:

```
        set_name.update(iterable)
```

```
   my_set = {1, 2, 3}

   my_set.update([4, 5, 6])

   print("After update:", my_set)
```

remove() - Removes a specified element from the set. Raises an error if the element is not found.

Syntax:

Example:

```
   my_set = {1, 2, 3, 4}

   my_set.remove(3)

   print("After remove:", my_set)
```

discard() - Removes a specified element from the set. Does not raise an error if the element is not found.

Syntax:

```
        set_name.discard(element)
```

Example:

```
   my_set = {1, 2, 3, 4}

   my_set.discard(5)  # No error

   print("After discard:", my_set)
```

pop() - Removes and returns an arbitrary element from the set.

Syntax:

```
        set_name.pop()
```

Example:

```
  my_set = {10, 20, 30, 40}

   popped_element = my_set.pop()
```

```
print("Popped Element:", popped_element)

print("After pop:", my_set)
```

clear() - Removes all elements from the set.

Syntax:

```
set_name.clear()
```

Example:

```
my_set = {1, 2, 3, 4, 5}

my_set.clear()

print("After clear:", my_set)
```

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

Syntax:

```
set1.union(set2)
```

Example:

```
set1 = {1, 2, 3}

set2 = {3, 4, 5}

union_set = set1.union(set2)

print("Union:", union_set)
```

intersection() - Returns a new set containing only common elements of both sets.

Syntax:

```
set1.intersection(set2)
```

Example:

```
set1 = {1, 2, 3}

set2 = {3, 4, 5}

intersect_set = set1.intersection(set2)
```

```
print("Intersection:", intersect_set)
```

difference() - Returns a new set with elements present in the first set but not in the second.

Syntax:

```
set1.difference(set2)
```

Example:

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

set2 = {3, 4, 5, 6}

diff_set = set1.difference(set2)

print("Difference:", diff_set)
```

symmetric_difference() - Returns elements that are in either set, but not in both.

Syntax:

```
set1.symmetric_difference(set2)
```

Example:

```
set1 = {1, 2, 3}

set2 = {3, 4, 5}

sym_diff_set = set1.symmetric_difference(set2)

print("Symmetric Difference:", sym_diff_set)
```

issubset() - Checks if all elements of `set1` are in `set2`.

Syntax:

```
set1.issubset(set2)
```

Example:

```
set1 = {1, 2}

set2 = {1, 2, 3, 4}
```

```
print("Is Subset:", set1.issubset(set2))
```

issuperset() - Checks if `set1` contains all elements of `set2`.

Syntax:

```
set1.issuperset(set2)
```

Example:

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

set2 = {2, 3}

print("Is Superset:", set1.issuperset(set2))
```

isdisjoint() - Checks if `set1` and `set2` have no common elements.

Syntax:

```
set1.isdisjoint(set2)
```

Example:

```
set1 = {1, 2, 3}

set2 = {4, 5, 6}

print("Is Disjoint:", set1.isdisjoint(set2))
```