Python - Set

A set is a collection of data types in Python, same as the list and tuple. However, it is not an ordered collection of objects. The set is a Python implementation of the set in Mathematics. A set object has suitable methods to perform mathematical set operations like union, intersection, difference, etc.

A set object contains one or more items, not necessarily of the same type, which are separated by comma and enclosed in curly brackets {}.

```
Syntax:
set = {value1, value2, value3,...valueN}
```

The following defines a set object.

```
>>> S1={1, "Bill", 75.50}
```

A set doesn't store duplicate objects. Even if an object is added more than once inside the curly brackets, only one copy is held in the set object. Hence, indexing and slicing operations cannot be done on a set object.

```
>>> S1={1, 2, 2, 3, 4, 4, 5, 5}
>>> S1
{1, 2, 3, 4, 5}
```

set() function

Python has an in-built function `set()`, using which a set object can be constructed out of any sequence such as a string, list or a tuple object.

```
>>> s1=set("Python")
>>> s1
```

```
{'t', 'h', 'o', 'n', 'P', 'y'}
>>> s2=set([45,67,87,36, 55])
>>> s2
{67, 36, 45, 87, 55}
>>> s3=set((10,25,15))
>>> s3
{25, 10, 15}
```

The order of elements in the set is not necessarily the same as the order given at the time of assignment. Python optimizes the structure of a set for performing operations over it, as defined in mathematics.

Only immutable (and hashable) objects can be a part of a set object. Numbers (integer, float, as well as complex), strings, and tuple objects are accepted, but list and dictionary objects are not.

```
>>> S1={(10,10), 10,20}
>>> S1
{10, 20, (10, 10)}
>>> S2={[10,10], 10,20}
TypeError: unhashable type: 'list'
```

In the above example, `(10,10)` is a tuple, hence it becomes part of the set. However, `[10,10]` is a list, hence an error message is displayed saying that the list is unhashable. (Hashing is a mechanism in computer science which enables quicker search of objects in the computer's memory.)

Even though mutable objects are not stored in a set, the set itself is a mutable object.

Set Operations

As mentioned earlier, the set data type in Python implements as the set defined in mathematics. Various set operations can be performed.

Operators |, &, - and ^ perform union, intersection, difference and symmetric difference operations, respectively. Each of these operators has a corresponding method associated with the built-in set class.

Operation	Operator/Method	Example

The union of two sets is a set of all elements from both the collections.	|	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1|s2
		{1, 2, 3, 4, 5, 6, 7, 8}

	union()	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1.union(s2)
		{1, 2, 3, 4, 5, 6, 7, 8}
		>>> s2.union(s1)
		{1, 2, 3, 4, 5, 6, 7, 8}

The intersection of two sets is a set containing elements common to both collections.	&	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1&s2
		{4, 5}
		>>> s2&s1
		{4, 5}

	intersection()	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1.intersection(s2)
		{4, 5}
		>>> s2.intersection(s1)
		{4, 5}

The difference of two sets results in a set containing elements only in the first set,	-	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1-s2
		{1, 2, 3}
		>>> s2-s1
		{8, 6, 7}

but not in the second set.	difference()	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1.difference(s2)
		{1, 2, 3}
		>>> s2.difference(s1)
		{8, 6, 7}

Symmetric Difference: the result of symmetric difference is a set consisting of elements in both sets, excluding the common elements.	^	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>> s1^s2
		{1, 2, 3, 6, 7, 8}
		>>> s2^s1
		{1, 2, 3, 6, 7, 8}

	Symmetric_difference()	>>> s1={1,2,3,4,5}
		>>> s2={4,5,6,7,8}
		>>>
		s1.symmetric_difference(s2)
		{1, 2, 3, 6, 7, 8}
		>>>
		s2.symmetric_difference(s1)
		{1, 2, 3, 6, 7, 8}

Built-in Set Methods

add()

Adds a new element in the set object.

```
>>> S1
{'Java', 'Python', 'C++'}
>>> S1.add("Perl")
>>> S1
{'Java', 'Python', 'Perl', 'C++'}
```

update()

Adds multiple items from a list or a tuple.

```
>>> S1={"Python", "Java", "C++"}
>>> S1.update(["C", "Basic"])
>>> S1
{'C++', 'Java', 'Python', 'Basic', 'C'}
>>> S1.update(("Ruby", "PHP"))
>>> S1
{'C++', 'Ruby', 'Java', 'PHP', 'Python', 'Basic', 'C'}
```

clear()

Removes the contents of set object and results in an empty set.

```
>>> S1.clear()
>>> S1
set()
```

copy()

Creates a copy of the set object.

```
>>> S1={"Python", "Java", "C++"}
>>> S2=S1.copy()
>>> S2
{'Java', 'Python', 'C++'}
```

discard()

Returns a set after removing an item from it. No changes are done if the item is not present.

```
>>> S1={"Python", "Java", "C++"}
>>> S1.discard("Java")
>>> S1
{'Python', 'C++'}
```

```
>>> S1={"Python", "Java", "C++"}
>>> S1.discard("SQL")
>>> S1
{'Java', 'Python', 'C++'}
```

remove()

Returns a set after removing an item from it. Results in an error if the item is not present.

```
>>> S1={"Python", "Java", "C++"}
>>> S1.remove("C++")
>>> S1
{'Java', 'Python'}
>>> S1={"Python", "Java", "C++"}
>>> S1.remove("SQL")
KeyError: 'SQL'
```