What is a set in Python?

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

How to create a set?

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function `set()`.

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

```
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Example -2

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Output

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}
<class 'set'>
looping through the set elements ...
Friday
Tuesday
Monday
Saturday
Thursday
Sunday
Wednesday
```

Example 2: using set() method

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
   print(i)
```

Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the `set()` function without any argument.

```
# initialize a with {}
a = {}

# check data type of a
# Output: <class 'dict'>
print(type(a))

# initialize a with set()
a = set()

# check data type of a
# Output: <class 'set'>
print(type(a))
```

How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning.

We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the `add()` method and multiple elements using the `update()` method. The `update()` method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4,5], {1,6,8})
print(my_set)
```

```
{1, 3}
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 8}
```

How to remove elements from a set?

A particular item can be removed from set using methods, `discard()` and `remove()`.

The only difference between the two is that, while using `discard()` if the item does not exist in the set, it remains unchanged. But `remove()` will raise an error in such condition.

The following example will illustrate this.

```
# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)

# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)

# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)

# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)

# remove an element
# not present in my_set
# If you uncomment line 27,
# you will get an error.
# Output: KeyError: 2

#my_set.remove(2)
```

Similarly, we can remove and return an item using the `pop()` method.

Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.

We can also remove all items from a set using `clear()`.

```
# initialize my_set
# Output: set of unique elements
my_set = set("HelloWorld")
print(my_set)

# pop an element
# Output: random element
print(my_set.pop())

# pop another element
# Output: random element
my_set.pop()
print(my_set)

# clear my_set
#Output: set()
my_set.clear()
print(my_set)
```

Difference between discard() and remove()

Example -2

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving items through discard() method...");
Months.discard("Feb"); #will not give an error although the key feb is not available in the set
print("\nprinting the modified set...")
print(Months)
print("\nRemoving items through remove() method...");
Months.remove("Jan") #will give an error as the key jan is not available in the set.
print("\nPrinting the modified set...")
print(Months)
```

Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods

Consider the following example to calculate the union of two sets.

Example 1 : using union | operator

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1|Days2) #printing the union of the sets
```

Output

```
{'Friday', 'Sunday', 'Saturday', 'Tuesday', 'Wednesday', 'Monday', 'Thursday'}
```

Python also provides the **union()** method which can also be used to calculate the union of two sets. Consider the following example.

Example 2: using union() method

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1.union(Days2)) #printing the union of the sets
```

Intersection of two sets

The & (intersection) operator is used to calculate the intersection of the two sets in python. The intersection of the two sets are given as the set of the elements that common in both sets.

Consider the following example.

Example 1: using & operator

```
set1 = {"Ayush","John", "David", "Martin"}
set2 = {"Steve","Milan","David", "Martin"}
print(set1&set2) #prints the intersection of the two sets
```

Output

```
{'Martin', 'David'}
```

Example 2: using intersection() method

```
set1 = {"Ayush","John", "David", "Martin"}
set2 = {"Steave","Milan","David", "Martin"}
print(set1.intersection(set2)) #prints the intersection of the two sets
```

The intersection_update() method

The intersection_update() method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).

The Intersection_update() method is different from intersection() method since it modifies the original set by removing the unwanted items, on the other hand, intersection() method returns a new set.

Consider the following example.

```
a = {"ayush", "bob", "castle"}
b = {"castle", "dude", "emyway"}
c = {"fuson", "gaurav", "castle"}

a.intersection_update(b, c)

print(a)
```

```
{'castle'}
```

Difference of two sets

The difference of two sets can be calculated by using the subtraction (-) operator. The resulting set will be obtained by removing all the elements from set 1 that are present in set 2.

Example 1 : using subtraction (-) operator or difference() method

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1-Days2) #{"Wednesday", "Thursday" will be printed}
```

```
{'Thursday', 'Wednesday'}
```

Set comparisons

Python allows us to use the comparison operators i.e., <, >, <=, >= , == with the sets by using which we can check whether a set is subset, superset, or equivalent to other set. The boolean true or false is returned depending upon the items present inside the sets.

Consider the following example.

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday"}
Days3 = {"Monday", "Tuesday", "Friday"}

#Days1 is the superset of Days2 hence it will print true.
print (Days1>Days2)

#prints false since Days1 is not the subset of Days2
print (Days1<Days2)

#prints false since Days2 and Days3 are not equivalent
print (Days2 == Days3)
```

Output

```
True
False
False
```

FrozenSets

The frozen sets are the immutable form of the normal sets, i.e., the items of the frozen set can not be changed and therefore it can be used as a key in dictionary.

The elements of the frozen set can not be changed after the creation. We cannot change or append the content of the frozen sets by using the methods like add() or remove().

```
Frozenset = frozenset([1,2,3,4,5])
print(type(Frozenset))
print("\nprinting the content of frozen set...")
for i in Frozenset:
    print(i);
Frozenset.add(6) #gives an error since we cannot change the content of Frozenset after creation
```

```
<class 'frozenset'>

printing the content of frozen set...
1
2
3
4
5
Traceback (most recent call last):
  File "set.py", line 6, in <module>
    Frozenset.add(6) #gives an error since we can change the content of Frozenset after creation
AttributeError: 'frozenset' object has no attribute 'add'
```

Python Built-in set methods

SN	Method	Description
1	add(item)	It adds an item to the set. It has no effect if the item is already present in the set.
2	clear()	It deletes all the items from the set.
3	copy()	It returns a shallow copy of the set.
4	difference_update(....)	It modifies this set by removing all the items that are also present in the specified sets.
5	discard(item)	It removes the specified item from the set.
6	intersection()	It returns a new set that contains only the common elements of both the sets. (all the sets if more than two are specified).
7	intersection_update(....)	It removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).
8	Isdisjoint(....)	Return True if two sets have a null intersection.
9	Issubset(....)	Report whether another set contains this set.
10	Issuperset(....)	Report whether this set contains another set.
11	pop()	Remove and return an arbitrary set element that is the last element of the set. Raises KeyError if the set is empty.
12	remove(item)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
13	symmetric_difference(....)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
14	symmetric_difference_update(....)	Update a set with the symmetric difference of itself and another.
15	union(....)	Return the union of sets as a new set. (i.e. all elements that are in either set.)
16	update()	Update a set with the union of itself and others.