Python Set

The set in python can be defined as the unordered collection of various items enclosed within the curly braces. The elements of the set can not be duplicate. The elements of the python set must be immutable.

Unlike other collections in python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together or we can get the list of elements by looping through the set.

Creating a set

The set can be created by enclosing the comma separated items with the curly braces. Python also provides the set method which can be used to create the set by the passed sequence.

Example 1: using curly braces

```
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)
```

Output:

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

Python Set operations

In the previous example, we have discussed about how the set is created in python. However, we can perform various mathematical operations on python sets like union, intersection, difference, etc.

Adding items to the set

Python provides the add() method which can be used to add some particular item to the set. Consider the following example.

Example:

```
1. Months = set(["January","February", "March", "April", "May", "June"])
2. print("\nprinting the original set ... ")
3. print(Months)
4. print("\nAdding other months to the set...");
5. Months.add("July");
6. Months.add("August");
7. print("\nPrinting the modified set...");
8. print(Months)
9. print("\nlooping through the set elements ... ")
10. for i in Months:
11.     print(i)
```

Output:

```
printing the original set ...
{'February', 'May', 'April', 'March', 'June', 'January'}

Adding other months to the set...

Printing the modified set...
{'February', 'July', 'May', 'April', 'March', 'August', 'June', 'January'}

looping through the set elements ...
February
```

```
July
May
April
March
August
June
January
```

To add more than one item in the set, Python provides the **update()** method.

Consider the following example.

Example

```
1. Months = set(["January","February", "March", "April", "May", "June"])
2. print("\nprinting the original set ... ")
3. print(Months)
4. print("\nupdating the original set ... ")
5. Months.update(["July","August","September","October"]);
6. print("\nprinting the modified set ... ")
7. print(Months);
```

Output:

```
printing the original set ...
{'January', 'February', 'April', 'May', 'June', 'March'}

updating the original set ...

printing the modified set ...
{'January', 'February', 'April', 'August', 'October', 'May', 'June', 'July', 'September', 'March'}
```

Removing items from the set

Python provides **discard()** method which can be used to remove the items from the set.

Consider the following example.

Example

```
1. Months = set(["January","February", "March", "April", "May", "June"])
2. print("\nprinting the original set ... ")
3. print(Months)
4. print("\nRemoving some months from the set...");
5. Months.discard("January");
6. Months.discard("May");
7. print("\nPrinting the modified set...");
8. print(Months)
9. print("\nlooping through the set elements ... ")
```

```
10. for i in Months:
11.     print(i)
```

Output:

```
printing the original set ...
{'February', 'January', 'March', 'April', 'June', 'May'}

Removing some months from the set...

Printing the modified set...
{'February', 'March', 'April', 'June'}

looping through the set elements ...
February
March
April
June
```

Python also provide the remove() method to remove the items from the set. Consider the following example to remove the items using remove() method.

Example

```
1. Months = set(["January","February", "March", "April", "May", "June"])
2. print("\nprinting the original set ... ")
3. print(Months)
4. print("\nRemoving some months from the set...");
5. Months.remove("January");
6. Months.remove("May");
7. print("\nPrinting the modified set...");
8. print(Months)
```

Output:

```
printing the original set ...
{'February', 'June', 'April', 'May', 'January', 'March'}

Removing some months from the set...

Printing the modified set...
{'February', 'June', 'April', 'March'}
```

We can also use the pop() method to remove the item. However, this method will always remove the last item.

Consider the following example to remove the last item from the set.

```
1. Months = set(["January","February", "March", "April", "May", "June"])
2. print("\nprinting the original set ... ")
3. print(Months)
```

```
print("\nRemoving some months from the set...");
Months.pop();
Months.pop();
print("\nPrinting the modified set...");
print(Months)
```

Output:

```
printing the original set ...
{'June', 'January', 'May', 'April', 'February', 'March'}

Removing some months from the set...

Printing the modified set...
{'May', 'April', 'February', 'March'}
```

Python provides the clear() method to remove all the items from the set.

Consider the following example.

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving all the items from the set...");
Months.clear()
print("\nPrinting the modified set...")
print(Months)
```

Output:

```
printing the original set ...
{'January', 'May', 'June', 'April', 'March', 'February'}

Removing all the items from the set...

Printing the modified set...
set()
```

Difference between discard() and remove()

Despite the fact that discard() and remove() method both perform the same task, There is one main difference between discard() and remove().

If the key to be deleted from the set using discard() doesn't exist in the set, the python will not give the error. The program maintains its control flow.

On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the python will give the error.

Consider the following example.

Example

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

Output:

```
printing the original set ...
{'March', 'January', 'April', 'June', 'February', 'May'}

Removing items through discard() method...

printing the modified set...
{'March', 'January', 'April', 'June', 'February', 'May'}

Removing items through remove() method...
Traceback (most recent call last):
  File "set.py", line 9, in
    Months.remove("Jan")
KeyError: 'Jan'
```

Union of two Sets

The union of two sets are calculated by using the or (|) operator. The union of the two sets contains the all the items that are present in both the sets.

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

Example 1 : using union | operator

```
1. Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
2. Days2 = {"Friday","Saturday","Sunday"}
3. 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
```

Output:

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

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
```

Output:

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

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)
```

Output:

```
{'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.

Consider the following example.

Example 1 : using subtraction (-) operator

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

Output:

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

Example 2 : using difference() method

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1.difference(Days2)) # prints the difference of the two sets Days1 and Days2
```

Output:

```
{'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().

The frozenset() method is used to create the frozenset object. The iterable sequence is passed into this method which is converted into the frozen set as a return type of the method.

Consider the following example to create the frozen set.

```
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
```

Output:

```
<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'
```

Frozenset for the dictionary

If we pass the dictionary as the sequence inside the frozenset() method, it will take only the keys from the dictionary and returns a frozenset that contains the key of the dictionary as its elements.

Consider the following example.

```
1. Dictionary = {"Name":"John", "Country":"USA", "ID":101}
2. print(type(Dictionary))
3. Frozenset = frozenset(Dictionary); #Frozenset will contain the keys of the dictionary
4. print(type(Frozenset))
5. for i in Frozenset:
6.     print(i)
```

Output:

```
<class 'dict'>
<class 'frozenset'>
Name
Country
ID
```

Python Built-in set methods

Python contains the following methods to be used with the sets.

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.