Methods of set and dictionary

01 August 2024 01:00 PM

Built-in method of sets:

1. copy( )———> is used for performing shallow copy
2. clear( )———> is used for deleting all the elements of given set

Example:

```
>>> s={11,22,33,44,55}
>>> scl=s.copy()
>>> id(s)
4379587776
>>> id(scl)
4379589568
>>> scl.clear()
>>> scl
set()
>>> s
{33, 22, 55, 11, 44}
```

Insertion method of set:

add( ) ———> it is used for adding the elements into set randomly

Syntax:

add(value)

note:

list, set, dictionary should not be used as value

Example:

```
>>> s ={33,22, 55, 11,44}
>>> s.add('hai')
>>> s
{33, 'hai', 22, 55, 11, 44}
>>> s.add((34,45,56))
>>> s
{33, 'hai', 22, 55, 11, 44, (34, 45, 56)}
```

Deletion methods of sets:

pop( )	remove( )	discard( )
Syntax: pop( )	Syntax: remove(value)	Syntax: discard(value)
1. It is used for deleting the first value from the defined set.	1. It is used for deleting the elements on given value	1. It is used for deleting the elements on given value
2. It will not take any arguments 3. If set is empty then pop gives key error	2. If value is present then remove( ) will delete that element else it will through the error	2. If value is present then discard( ) will delete that element else it will not perform any operation

Example:

```
>>> s ={11,33,56,89,23,19}
>>> s
{33, 19, 23, 56, 89, 11}
>>> s.pop()
33
>>> s.pop()
19
>>> s
{23, 56, 89, 11}
>>> s.remove(56)
>>> s
{23, 89, 11}
>>> s.remove(56)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    s.remove(56)
KeyError: 56
>>> s.discard(23)
>>> s
{89, 11}
>>> s.discard(23)
>>> s
{89, 11}
```

union( )	intersection( )	difference( )
Syntax: Basset.union(set1,set2,……)	Syntax: Basset.intersection(set1,set2,…)	Syntax: Basset.difference(set1,set2,……)
It is used for returning all the elements from the specified set	It is used for returning the only common elements from the specified set	It is used for returning the un-common elements form the baseSet.

Note:

By above operations any of the specified sets will not get modified.

Example:

```
>>> s = {11,22,33,44}
>>> s1= {44,55,66,77}
>>> s2={22,88,99}
>>> s.union(s1)
    {33, 66, 11, 44, 77, 22, 55}
>>> s.union(s2)
    {33, 99, 22, 88, 11, 44}
>>> s.union(s1,s2)
    {33, 66, 99, 11, 44, 77, 22, 55, 88}
>>> s.intersection(s1)
    {44}
>>> s.intersection(s2)
    {22}
>>> s.intersection(s1,s2)
    set()
>>> s.difference(s1)
    {33, 11, 22}
>>> s1.difference(s)
    {66, 77, 55}
```

methods	syntax	Functionality
update( )	Basset.update(set1,set2,……)	It will perform union operation and update the output into base set
intersection_update( )	Basset.intersection_update(set1,set2,……)	It will perform intersection operation and update the output into base set
difference_update( )	Basset.difference_update(set1,set2,……)	It will perform difference operation and update the output into base set
symmetric_difference_update( )	Basset.symmetric_difference_update(set1)	It will perform symmetric_difference operation and update the output into base set

Example:

```
>>> s = {33,11,44,22}
>>> s1 = {56,89,44}
>>> s2 = {34,53,22}
>>> s.update(s1)
>>> s
    {33, 22, 56, 89, 11, 44}
>>> s.update(s1,s2)
>>> s
    {33, 34, 11, 44, 53, 22, 56, 89}
>>> s.intersection_update(s1)
>>> s
    {56, 89, 44}
>>> s3 = {34,90,89}
>>> s.difference_update(s3)
>>> s
    {56, 44}
>>> s4 = {90,67,56}
>>> s3.symmetric_difference_update(s4)
>>> s
    {56, 44}
```

issuperset( )	issubsset( )	isdisjoint( )
Returns true if the base set is the superior of specified set else it return false	Returns true if the base set is the inferior of specified set else it return false	Returns true if the of specified set has only un common element else it return false
Syntax: Basset.issuperset(set1)	Syntax: Basset.issubset(set1)	Syntax: Basset.isdisjoint(set1)

example:

```
>>> s = {11,22}
>>> s1 = {11,22,33}
>>> s2 ={11,44}
>>> s3 = {11,33}
>>> s1.issuperset(s)
    True
>>> s2.issuperset(s)
    False
>>> s3.issuperset(s)
    False
>>> {11,22}.issuperset(s)
    True
>>> s.issubset(s1)
    True
>>> s.issubset(s2)
    False
>>> s.issubset({11,22})
    True
>>> s.disjoint(s1)
```

Built - in methods of dictionary:

keys( )	values( )	items( )
Syntax: keys( )	Syntax: values( )	Syntax: items( )
It is used for getting the keys of specified dictionary	It is used for getting the values of specified dictionary	It is used for getting the both keys and values of specified dictionary

example:

```
>>> d = {'Name':'Suvrat','Age':22}
>>> d
    {'Name': 'Suvrat', 'Age': 22}
>>> d.keys()
    dict_keys(['Name', 'Age'])
>>> d.values()
    dict_values(['Suvrat', 22])
>>> d.items()
    dict_items([('Name', 'Suvrat'), ('Age', 22)])
```

1. copy( ) ——> it is used for performing shallow copy
2. clear( )——-> it will delete only the elements from the given dictionary

```
>>> d = {'name':'suvrat','age':22}
>>> d1 = d.copy()
>>> id(d)
    4342121600
>>> id(d1)
    4354946496
>>> d1.clear()
>>> d1
    {}
>>> d
    {'name': 'suvrat', 'age': 22}
```

get( )	setdefault( )
Syntax: get(keynote,[defaultvalue])	Syntax: setdefault(keynote,[defaultvalue])
If the specified key is present then get will display the value of specified key else get will return the default value and it will not update dictionary	If the specified key is present than setdefault will display the value of specified key else set default will return default value and update the dictionary with specified key and default value

example:

```
>>> d = {'name':'suvrat','age':22}
>>> d
    {'name': 'suvrat', 'age': 22}
>>> d.get('nmae')
>>> d.get('name')
    'suvrat'
>>> d.setdefault('name')
    'suvrat'
>>> d.setdefault('Nmae','Prakash')
    'Prakash'
>>> d
    {'name': 'suvrat', 'age': 22, 'Nmae': 'Prakash'}
```

Update method:

It is used for updating the actual dictionary with multiple key and value pairs at a time

Note:

It is used for merging dictionaries

We can provide pairs of data as value of update method

Syntax:

base dictionary.update({'key1:'value1','key2':"value2'……})

If specified key I present then it will update the value

If key I snot present then it will create a new key value pais

Example:

```
>>> d ={'name':'ashu'}
>>> d.update({'mobile':90352134,'age':3})
>>> d
    {'name': 'ashu', 'mobile': 90352134, 'age': 3}
>>> d.update([('age',4),('gender','Male')])
>>> d
    {'name': 'ashu', 'mobile': 90352134, 'age': 4, 'gender': 'Male'}
```

Deletion methods:

pop( )	popitem( )
Syntax : pop(keyname)	Syntax: popitem( )
1. It is used for deleting the key value pairs based on specified key 2. If key is present then it will delete else it throws an key error	1. It is used for deleting last key and value pairs by default 2. Output format of popitem is tuple 3. If dictionary is empty it throws an key error

Example:

```
>>> d ={'name':'suvrat','age':4}
>>> d
    {'name': 'suvrat', 'age': 4}
>>> d.pop('age')
    4
>>> d
    {'name': 'suvrat'}
>>> d.popitem()
    ('name', 'suvrat')
>>> d
    {}
```

Fromkeys:

Syntax:

fromkeys(CDT,[defaultvalues])

It is used for extracting each and every element of CDT and represent them as key and default values as a value for all the keys

```
>>> {}.fromkeys('bye',10)
    {'b': 10, 'y': 10, 'e': 10}
>>> d={}.fromkeys(['hello','hai','bye'],10)
>>> d
    {'hello': 10, 'hai': 10, 'bye': 10}
>>> {}.fromkeys('bye')
    {'b': None, 'y': None, 'e': None}
```