Set
set is unordered collection, where insertion order is not preserved
the order of insertion is not same.
Set does not allows duplicates elements/values

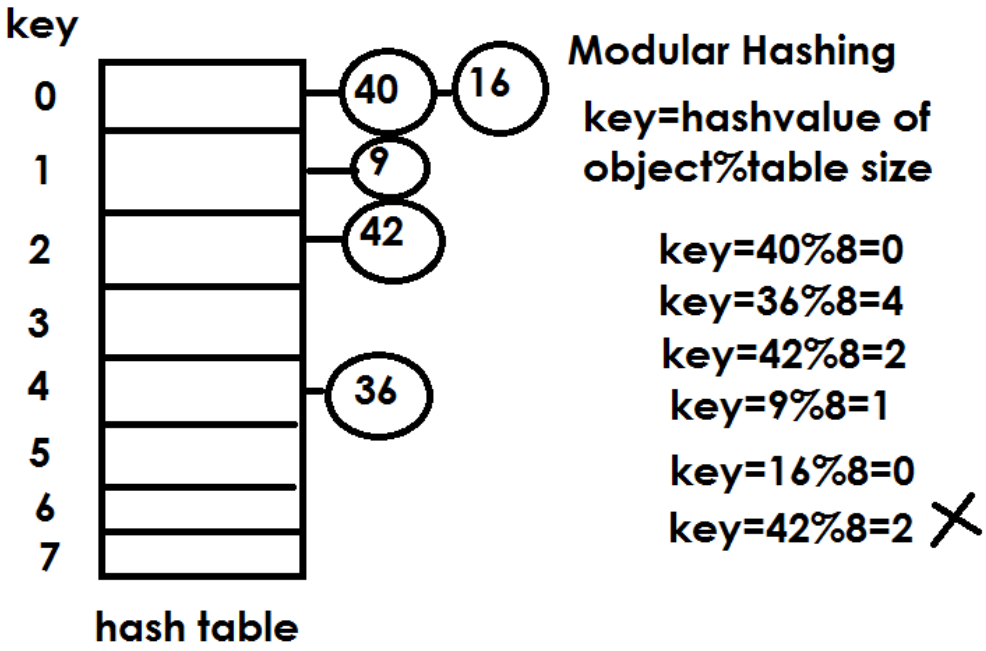
A *set* object is an unordered collection of distinct hashable objects.
Common uses include membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference.
Set uses hashing data structure for organizing of objects
According hashing data structure there is a hash table and each location in hash table is identified with key

Set is a mutable collection, after creating set we can add and delete.

What is hash value or hash code?
Hash value or hash code is an integer value, which is used by hash based data structures for generating key
According to hashing if two objects are equal, it should generate same hash value or hash code
hash(): it is predefined function in python, which return hash value or hash code of an object

```
>>> a=15
>>> b=15
>>> a==b
True
>>> hash(a)
15
>>> hash(b)
15
>>> s1="abc"
>>> s2="abc"
>>> s1==s2
True
>>> hash(s1)
6517954956523061052
>>> hash(s2)
6517954956523061052
>>>
```

```
>>> f2=1.5
>>> f1==f2
True
>>> hash(f1)
1152921504606846977
>>> hash(f2)
1152921504606846977
>>> id(f1)
586914464720
>>> id(f2)
586914464144
>>>
```



“set” class/data type is used to create set object

How to create set?

Sets can be created by several means:

- Use a comma-separated list of elements within braces: {'jack', 'sjoerd'}
- Use a set comprehension: {c for c in 'abracadabra' if c not in 'abc'}
- Use the type constructor: set(), set('foobar'), set(['a', 'b', 'foo'])

Example:

```
>>> set1={}
>>> type(set1)
```

```
<class 'dict'>
>>> set2=set()
>>> type(set2)
<class 'set'>
>>>
```

Empty set cannot created using empty curly braces, because curly braces are also used for creating dictionary.

```
>>> set2=set()
>>> type(set2)
<class 'set'>
>>> set1={10}
>>> print(set1)
{10}
>>> print(type(set1))
<class 'set'>
>>> set2={10,10,10,10,10}
>>> print(set2)
{10}
>>> print(type(set2))
<class 'set'>
>>> set3={10,20,30,40,50}
>>> print(set3)
{50, 20, 40, 10, 30}
>>> set4={10,1.5,"naresh",1+2j}
>>> print(set4)
{1.5, 10, 'naresh', (1+2j)}
>>> print(type(set4))
<class 'set'>
>>>
```

Use the type constructor: set(), set('foobar'), set(['a', 'b', 'foo'])

```
>>> set1=set()
>>> set2=set(range(10,110,10))
>>> print(set1)
set()
>>> print(set2)
{100, 70, 40, 10, 80, 50, 20, 90, 60, 30}
>>> list1=[10,20,30,40,50]
```

```
>>> set1=set(list1)
>>> print(set1)
{40, 10, 50, 20, 30}
>>> list2=[1,2,3,4,5,1,2,3,4,5]
>>> set2=set(list2)
>>> print(list2)
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
>>> print(set2)
{1, 2, 3, 4, 5}
>>> set3=set("PYTHON")
>>> print(set3)
{'O', 'H', 'N', 'T', 'Y', 'P'}
>>> set4=set("JAVA")
>>> print(set4)
{'A', 'V', 'J'}
>>>
```

How to read values/items from set?

```
>>> s1={10,20,30,40,50}
>>> s1[0]
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    s1[0]
TypeError: 'set' object is not subscriptable
>>>
```

We can read values from set using,

1. For loop
2. Iterator
3. Enumerate

Example: reading elements/values from set using for loop

```
set1={10,20,30,40,50}
for x in set1:
    print(x)
```

Output:

```
50
20
```

40
10
30
>>>

Example: reading elements from set using iterator

```
set1={10,20,30,40,50}
i=iter(set1)
value1=next(i)
value2=next(i)
print(value1)
print(value2)
```

Output:

```
50
20
>>>
```

Example: reading elements from set using enumerate

```
set1={10,20,30,40,50}
e1=enumerate(set1)
t1=next(e1)
t2=next(e1)
print(t1)
print(t2)
```

Output:

```
(0, 50)
(1, 20)
>>>
```