Nested set
Set inside set is called nested set.

frozenset
frozenset is an immutable set. After creating frozenset changes cannot be done. The frozenset does not support mutable operations

1. Add()
2. Remove()
3. Discard()
4. Clear()
5. Update()
6. Difference_update()
7. …

What is use of frozenset?

1. Representing immutable set
2. Nested set

How to create frozenset?

1. **frozenset()** : this returns empty frozenset
2. **frozenset(iterable)** : this returns frozenset from existing iterable

```
>>> f1=frozenset()
>>> print(f1)
frozenset()
>>> f1.add(10)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    f1.add(10)
AttributeError: 'frozenset' object has no attribute 'add'
>>> f2=frozenset(range(10,110,10))
>>> print(f2)
frozenset({100, 70, 40, 10, 80, 50, 20, 90, 60, 30})
>>> f2.remove(10)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    f2.remove(10)
```

```
>>> A=frozenset(range(10,60,10))
>>> print(A)
frozenset({40, 10, 50, 20, 30})
>>> B=frozenset(range(60,100,10))
>>> print(B)
frozenset({80, 90, 60, 70})
>>> C=A.union(B)
>>> print(A,B,C,sep="\n")
frozenset({40, 10, 50, 20, 30})
frozenset({80, 90, 60, 70})
frozenset({70, 40, 10, 80, 50, 20, 90, 60, 30})
```

Example:

```
>>> A={frozenset(range(10,60,10)),frozenset(range(60,110,10))}
>>> print(A)
{frozenset({100, 70, 80, 90, 60}), frozenset({40, 10, 50, 20, 30})}
>>> for S in A:
...     for x in S:
...         print(x,end=' ')
...     print()
...
...
100 70 80 90 60
40 10 50 20 30
```

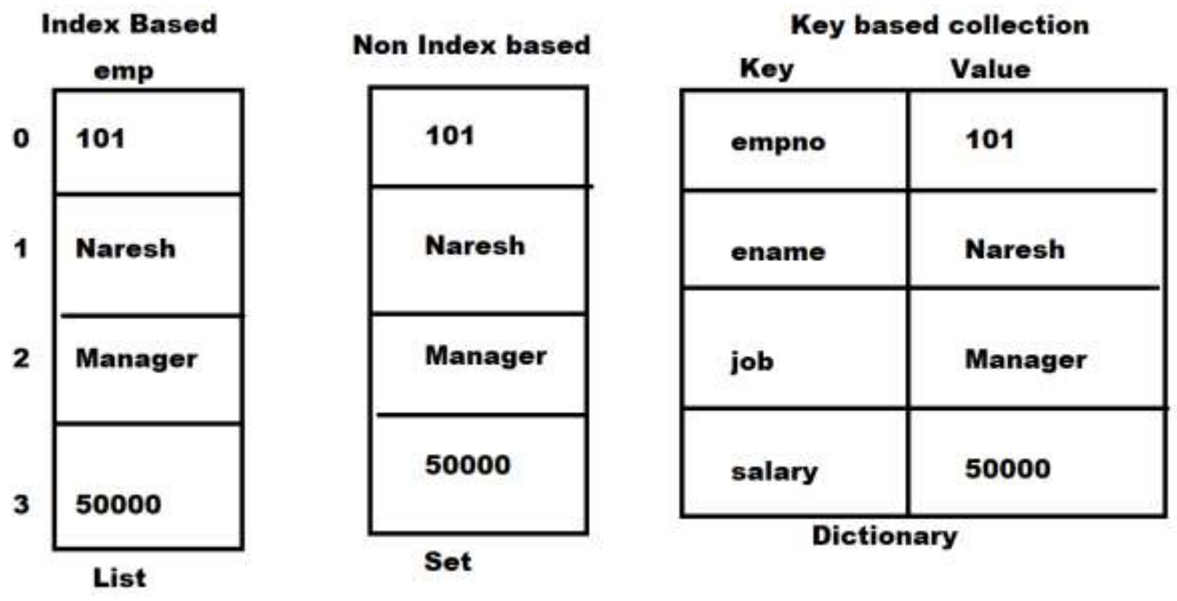
What is difference between list and set?

List	**Set**
List is ordered is a collection	set is unordered collection
List allows duplicates	Set does not allows duplicates
List support indexing and slicing	Set does not support indexing and slicing
List allows any type of objects	Set allows only immutable objects
In list data is organized in sequentially	In set data is organized using hashing data structure
In application development list is used to represent group	In application development set is used to represent group of

individual objects where duplicates are allowed and reading and writing is done sequentially and random.	individual objects where duplicates are not allowed and perform mathematical set operations.
List is created using []	Set is created using {}
“list” class or data type represents list object	“set” class or data type represents set object

Dictionary (mapping)

Dictionary mutable collection, after creating dictionary changes can be done.
Dictionary is mapping collection, where data is organized as key and value pair. Each value in dictionary is identified with key.
One key is mapped with one or more than one value.
Dictionary does not allows duplicate keys but allowed duplicate values.
Dictionary keys are immutable and values are mutable.
Dictionary keys are hashable objects/immutable objects.
In application developing to organize data as key and value pair use dictionary.



How to create dictionary?

1. Empty dictionary is created using empty curly braces {}

```
>>> d1={}
>>> print(type(d1),d1)
<class 'dict'> {}
```

2. Create a dictionary with items, where each item consist of key and value which is separated using :

Syntax: {key:value,key:value,key:value,….}

```
>>> d2={1:10,2:20,3:30,4:40,5:50}
>>> print(type(d2),d2)
<class 'dict'> {1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> person={'naresh':50,'suresh':40,'ramesh':20}
>>> print(person)
{'naresh': 50, 'suresh': 40, 'ramesh': 20}
>>> d3={1:10,1:20,1:30,1:40}
>>> print(d3)
{1: 40}
```

3. Creating dictionary using existing iterables/collections

Syntax1: dict()   : returns empty dictionary
Syntax2: dict(iterable) : convert existing iteable into dictionary

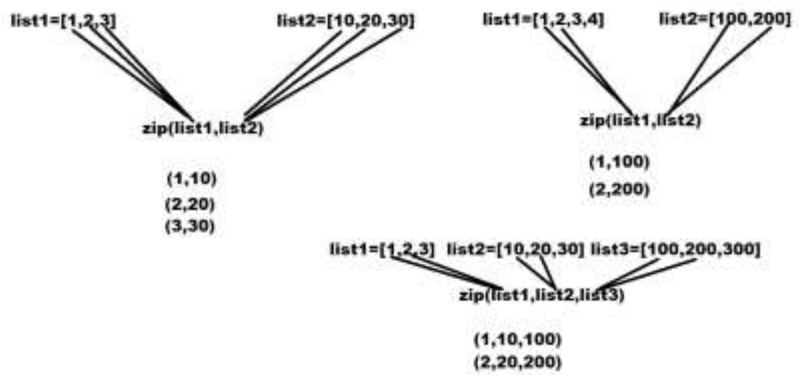
```
>>> d4=dict()
>>> print(d4,type(d4))
{} <class 'dict'>

>>> list1=[10,20,30,40,50]
>>> d5=dict(list1)
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    d5=dict(list1)
```

```
TypeError: cannot convert dictionary update sequence element #0
to a sequence
>>> list1=[(1,10),(2,20),(3,30),(4,40),(5,50)]
>>> d5=dict(list1)
>>> print(d5)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> list2=[10,20,30,40,50]
>>> e=enumerate(list2)
>>> d6=dict(e)
>>> print(d6)
{0: 10, 1: 20, 2: 30, 3: 40, 4: 50}
>>> list1=[1,2,3,4,5]
>>> list2=[10,20,30,40,50]
>>> list3=[(list1[i],list2[i]) for i in range(5)]
>>> print(list1,list2,list3,sep="\n")
[1, 2, 3, 4, 5]
[10, 20, 30, 40, 50]
[(1, 10), (2, 20), (3, 30), (4, 40), (5, 50)]
>>> d7=dict(list3)
>>> print(d7)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
```

zip(*iterables, strict=False)
Iterate over several iterables in parallel, producing tuples with an item from each one.

More formally: zip() returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument iterables.



```
>>> d8=dict(zip(range(1,6),range(10,60,10)))
>>> print(d8)
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50}
>>> d9=dict(zip("ABC","XYZ"))
>>> print(d9)
{'A': 'X', 'B': 'Y', 'C': 'Z'}
>>> d10=dict(zip("ABCD",range(65,69)))
>>> print(d10)
{'A': 65, 'B': 66, 'C': 67, 'D': 68}
>>> d11=dict(d10)
>>> print(d11)
{'A': 65, 'B': 66, 'C': 67, 'D': 68}
```

How to read content of dictionary?

1. Using key
2. Using for loop
3. Using dictionary methods
    a. Keys
    b. Values
    c. Items
    d. Get
    e. Setdefault
    f. Reversed

Using key
Dictionary is a key based and reading and writing is done using key

Syntax:
dictionary-name[key]

if key exists, returns its value
if key not exists, raise KeyError

Example:
```
# reading employees data using key

emp={'empno':101,
     'ename':'naresh',
     'job':'manager',
     'salary':50000}

print(emp['empno'],emp['ename'],emp['job'],emp['salary'])

emp_data={'empno':[101,102,103],
         'ename':['naresh','suresh','ramesh']}

print(emp_data['empno'],emp_data['ename'])

emp_info={101:['naresh',50000],
         102:['suresh',60000],
         103:['ramesh',70000]}

print(emp_info[101],emp_info[102],emp_info[103])
print(emp_info[101][0],emp_info[101][1])
print(emp_info[102][0],emp_info[102][1])
print(emp_info[103][0],emp_info[103][1])
```

Output

```
101 naresh manager 50000
[101, 102, 103] ['naresh', 'suresh', 'ramesh']
['naresh', 50000] ['suresh', 60000] ['ramesh', 70000]
naresh 50000
suresh 60000
ramesh 70000
```

using for loop