Syntax-6: list-name[start:stop:step]
Syntax-7: list-name[:stop:step]
Syntax-8: list-name[start::step]

Syntax-6: list-name[start:stop:step]

```
>>> list1=list(range(10,110,10))
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list2=list1[0:10:2]
>>> print(list2)
[10, 30, 50, 70, 90]
>>> list3=list1[2:7:2]
>>> print(list3)
[30, 50, 70]
>>> list4=list1[-5:-9:-2]
>>> print(list4)
[60, 40]
```

Syntax-7: list-name[:stop:step]

```
>>> list1=list(range(10,110,10))
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list2=list1[:6:1]
>>> print(list2)
[10, 20, 30, 40, 50, 60]
>>> list2=list1[:6:-1]
>>> print(list2)
[100, 90, 80]
>>> list3=list1[:-6:-1]
>>> print(list3)
[100, 90, 80, 70, 60]
```

Syntax-8: list-name[start::step]

```
list1=list(range(10,110,10))
print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```
list2=list1[2::2]
print(list2)
[30, 50, 70, 90]
list3=list1[-2::-2]
print(list3)
[90, 70, 50, 30, 10]
```

What is difference between indexing and slicing?

Indexing	**Slicing**
Using index we can read one value	Using slicing we can read more than one value
Index required only position of value	Slicing required 3 values 1. Start 2. Stop 3. Step
This return value	This return another sequence/list
If invalid index is given it raises IndexError	This never raise any error, it returns empty list

Slicing object

"slice" data type or class represent slice object.
"slice" object is reusable. Once slice object is created we can use to slice one or more than one sequence or list.

Syntax:
slice(start,stop,step)

Example:

```
s=slice(0,7)
list1=list(range(10,110,10))
print(list1)
list2=list1[s]
print(list2)
list3=list(range(100,200,10))
print(list3)
list4=list3[s]
print(list4)
```

Output:

[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[10, 20, 30, 40, 50, 60, 70]
[100, 110, 120, 130, 140, 150, 160, 170, 180, 190]
[100, 110, 120, 130, 140, 150, 160]

Iterator
Iterator object is used to read elements/items/values from collections (list,set,dictionary). Iterator allows to read in forward direction. Using iterator we can read values from collection but we cannot do any changes.

How to create iterator object?
Iter() is a predefined function, which return iterator object.

Syntax: iter(iterable)

Example:

```
>>> a=iter(list1)
>>> type(list1)
<class 'list'>
type(a)
<class 'list_iterator'>
>>> next(a)
10
>>> next(a)
20
>>> next(a)
30
>>> next(a)
40
>>> next(a)
50
>>> next(a)
60
>>> for value in a:
...     print(value)
...
...
70
```

80
90
100

next() is a predefined function, this function return next value generated by iterator object.

```
>>> namesList=['naresh','suresh','ramesh']
>>> a=iter(namesList)
>>> next(a)
'naresh'
>>> next(a)
'suresh'
>>> next(a)
'ramesh'
>>> next(a)
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    next(a)
StopIteration
```

Syntax: next(iterator)

enumerate

enumerate is similar to iterator
enumerate return two values

1. Value of collection
2. Count

This enumerate is used to convert list into dictionary
Enumerate return value and count as tuple.

Syntax:
enumerate(iterable,start=0)

```
>>> list1=[10,20,30,40,50]
>>> e1=enumerate(list1)
>>> next(e1)
(0, 10)
>>> next(e1)
(1, 20)
>>> next(e1)
(2, 30)
>>> next(e1)
(3, 40)
>>> next(e1)
(4, 50)
>>> next(e1)
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in <module>
    next(e1)
StopIteration
>>> e1=enumerate(list1,start=100)
>>> next(e1)
(100, 10)
>>> next(e1)
(101, 20)
>>> next(e1)
(102, 30)
>>> next(e1)
(103, 40)
```

What is difference between iteartor and enumerate?

Iterator object return value	Enumerator return count and value

Methods List