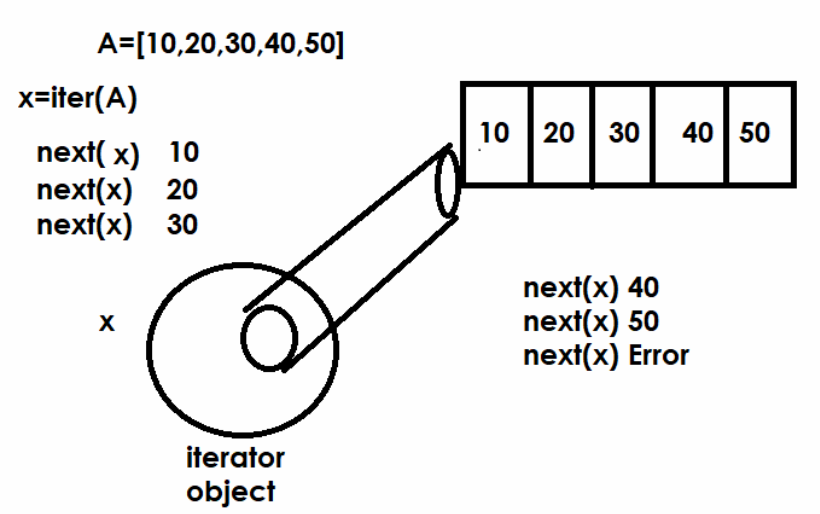
Iterator

Iterator object is used to iterate or read values from any collection or iterable.
Iterator is read only, it used for reading values but cannot change or update values.

Syntax: iter(Iterable)

In application development iterator is used for non index based collections (sets, mappings,..)



This iterator allows reading elements/values in forward direction.

```
>>> A=[10,20,30,40,50]
>>> x=iter(A)
>>> next(x)
10
>>> next(x)
20
>>> next(x)
30
>>> next(x)
40
>>> next(x)
50
>>> next(x)
```

```
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    next(x)
StopIteration
```

Iterator object can be given to for loop to iterate each time one value and perform operation.

```
>>> A=[10,20,30,40,50,60,70,80,90,100]
>>> y=iter(A)
>>> for value in y:
    print(value,end=' ')


10 20 30 40 50 60 70 80 90 100
```

Enumerate

Enumerate is an iterator, which returns 2 values

1. The value read from iterable or collection
2. Count (default count start at 0)

```
names=["naresh","suresh","ramesh","rajesh"]
x=enumerate(names,1)

next(x) --> (1,naresh)
next(x) --> (2,suresh)
next(x) --> (3,ramesh)
next(x) --> (4,rajesh)
next(x) --> Error
```

```
sales=[4500,5000,6000,9000,8000]

x=enumerate(sales,2010)

next(x) --> (2010,4500)
next(x) --> (2011,5000)
next(x) --> (2012,6000)
next(x) --> (2013,9000)
next(x) --> (2014,8000)
next(x) --> Error
```

```
>>> names=["naresh","ramesh","kishore","rajesh"]
>>> x=enumerate(names)
>>> next(x)
(0, 'naresh')
>>> next(x)
(1, 'ramesh')
```

```
>>> next(x)
(2, 'kishore')
>>> next(x)
(3, 'rajesh')
>>> next(x)
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    next(x)
StopIteration
>>> A=[10,20,30,40,50,60,70,80,90,100]
>>> y=enumerate(A,1)
>>> for p in y:
    print(p)

...
(1, 10)
(2, 20)
(3, 30)
(4, 40)
(5, 50)
(6, 60)
(7, 70)
(8, 80)
(9, 90)
(10, 100)
```

Mutable operation of list

List is a mutable sequence, after creating list changes can be done.

1. **append()**
2. **extend()**
3. **insert()**
4. **remove()**

5. **clear()**
6. **pop()**
7. **sort()**
8. **replace or update**
9. **using del keyword**

append() method

append is function or method of list data type or class.
This method is used to append or add element or item at the end of list.
This method is used to append or add single element or value

Syntax: list-name.append(value/item)

```
>>> A=[]
>>> print(A)
[]
>>> A.append(10)
>>> print(A)
[10]
>>> A.append(20)
>>> print(A)
[10, 20]
>>> A.append(30)
>>> print(A)
[10, 20, 30]
>>> A.append(40)
>>> print(A)
[10, 20, 30, 40]
>>> A.append(50)
>>> print(A)
[10, 20, 30, 40, 50]
>>> A.append(100,200)
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    A.append(100,200)
TypeError: list.append() takes exactly one argument (2 given)
```

Example:

```
# Write a program to append scores of "n" players
# into list and calculate total score

n=int(input("How many players ?"))
scores=[]

for i in range(n):
    s=int(input("Enter Score :"))
    scores.append(s)


print(f'Scores are {scores}')
total=0
for s in scores:
    total=total+s

print(f'Total Score {total}')
```

Output

```
How many players ?3
Enter Score :100
Enter Score :20
Enter Score :40
Scores are [100, 20, 40]
Total Score 160
```

Example

```
# Write a program to read "n" integers into list
# and count even,odd

n=int(input("Enter How Many Integer Values ?"))
A=[]

for _ in range(n):
    value=int(input("Enter any integer value"))
    A.append(value)

ec=0
oc=0
```

```
for value in A:
    if value%2==0:
        ec+=1
    else:
        oc+=1

print(f'List is {A}')
print(f'Even Count {ec}')
print(f'Odd Count {oc}')
```

Output

```
Enter How Many Integer Values ?10
Enter any integer value1
Enter any integer value2
Enter any integer value3
Enter any integer value4
Enter any integer value5
Enter any integer value6
Enter any integer value7
Enter any integer value8
Enter any integer value9
Enter any integer value10
List is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Even Count 5
Odd Count 5
```

Example:

```
# Write program from given list separate +ve and -ve

A=[10,20,-30,40,50,-70,-80,-90,55,66,25,53,67,87,-45,-33]
B=[]
C=[]

for value in A:
    if value>0:
        B.append(value)
    else:
        C.append(value)

print(f'List is {A}')
```

```
print(f'List with +ve numbers {B}')
print(f'List with -ve numbers {C}')
```

Output

```
List is [10, 20, -30, 40, 50, -70, -80, -90, 55, 66, 25, 53, 67, 87, -45, -33]
List with +ve numbers [10, 20, 40, 50, 55, 66, 25, 53, 67, 87]
List with -ve numbers [-30, -70, -80, -90, -45, -33]
```

extend () methods

This method is used to append more the one value.

Syntax: list-name.extend(iterable/collection)

```
>>> A=[]
>>> A.append(10)
>>> print(A)
[10]
>>> A.extend([20,30,40])
>>> print(A)
[10, 20, 30, 40]
>>> A.extend(range(50,80,10))
>>> print(A)
[10, 20, 30, 40, 50, 60, 70]
>>> A.extend("NIT")
>>> print(A)
[10, 20, 30, 40, 50, 60, 70, 'N', 'I', 'T']
>>> A.append("NIT")
>>> print(A)
[10, 20, 30, 40, 50, 60, 70, 'N', 'I', 'T', 'NIT']
>>> A.extend("NIT")
>>> print(A)
[10, 20, 30, 40, 50, 60, 70, 'N', 'I', 'T', 'NIT', 'N', 'I', 'T']
>>> A.extend(["python","java","oracle"])
>>> print(A)
[10, 20, 30, 40, 50, 60, 70, 'N', 'I', 'T', 'NIT', 'N', 'I', 'T', 'python', 'java',
'oracle']
```

Append multiple values using slicing operator

Without using extend method, we can append multiple values/items with help of slicing operator

Syntax: list-name[len(list-name):]=iterable

```
>>> A=[10,20,30]
>>> print(A)
[10, 20, 30]
>>> A[len(A):]=[40,50]
>>> print(A)
[10, 20, 30, 40, 50]
>>> A[len(A):]=[1.5,2.5,3.5]
>>> print(A)
[10, 20, 30, 40, 50, 1.5, 2.5, 3.5]
>>> A[len(A):]=["naresh","ramesh"]
>>> print(A)
[10, 20, 30, 40, 50, 1.5, 2.5, 3.5, 'naresh', 'ramesh']
>>> A[len(A):]="ABC"
>>> print(A)
[10, 20, 30, 40, 50, 1.5, 2.5, 3.5, 'naresh', 'ramesh', 'A', 'B', 'C']
```

Replacing values or updating values

List is a mutable collection, we can replace or update values of list
This replacing or updating values of list is done in 2 ways

1. using index
2. using slicing