Iterator is a python object which can be iterated upon. Python lists, tuples, dictionaries and sets are all examples of built-in iterators. Python **iterator object** must implement two methods, ***__iter__()*** and ***__next__()*** .

1. __iter__ : method that is called on initialization of an iterator. This should return an object that has a next or __next__ method.

2. __next__ : The iterator next method should return the next value for the iterable. When an iterator is used with a 'for in' loop, the for loop implicitly calls next() on the iterator object. This method should raise a StopIteration to signal the end of the iteration.

Iterating through iterators

Now, we will use these two methods to iterate through iterator:

```
list_of_numbers=[1,2,3,4]# get an iterator using iter()
iterator = iter(list_of_numbers)## iterate through it
using next()print(next(iterator))          #prints
1print(next(iterator))          #prints
2print(iterator.__next__())     #prints
3print(iterator.__next__())     #prints 4
```

```
next(iterator)
Output:
1
2
3
4
Traceback (most recent call last):
  File "/home/main.py", line 31, in <module>
    next(iterator)
StopIteration
```

Using ***iter()*** we can get an iterator.
Using ***next*** or ***__next__()*** we are fetching the next value in the list.

Last line will result in error because there is no more data to be returned.

In simplest way, we can use for loop to print the entire list:

```
list_of_numbers=[1,2,3,4]for i in
list_of_numbers:
    print(i)Output:
1
2
3
4
```

What does it mean? It means, for loop uses iterator internally and automatically will iterate through the list and return it.

how for loop executes internally ?

Syntax for ***for loop***:

```
for element in iterable:
    # coding part
```

Let’s see how exactly it executes:

1. ***for loop*** will create an object from the ***iterable()*** .
```
iterable_obj = iter(iterable)
```

2. then will go to an infinite loop where it calls next() method to get the next element and executes the for loop body with this value as shown below:

```
while True:
    try:
        element = next(iterable_obj)    #to get
the next item

    except StopIteration:
        # if StopIteration is raised, break from
loop
        break
```

After all elements visited, StopIteration will be raised and loop ends.

Creating an iterator

Here, we are calculating a square of numbers by creating our own iterator. For this you have to use only two methods, ***__iter__()*** and ***__next__()*** .

```
class square:

    """Class to implement an iterator
    of square of a number"""def __init__(self,x=0):
        self.x = x                   #x will be
assigned as 3def __iter__(self):
        self.n = 0                   #n is initialized
```

```
to 0
        return selfdef __next__(self):
        if self.n <= self.x:         #calculate square
if <=3
            result = self.n*self.n   #squaring a number
            self.n += 1              #increment number
            return result
        else:
            raise StopIteration     #if n>3, throw
exception

a=square(3)
i = iter(a)
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))Output:0
1
4
9
Traceback (most recent call last):
  File "/home/main.py", line 34, in <module>
    print(next(i))
  File "/home/main.py", line 26, in __next__
    raise StopIteration
StopIteration
```

Here, we are creating an ***__iter__()*** function to create an iterable object and then ***__next__()*** method is called to iterate over a list and return desired result.

If all the elements in the list are covered and if you tried to call ***__next__()*** , an exception will be raised since there is no element left to return in the list.

We can use ***for loop*** for above example as follows:

```
class square:
    """Class to implement an iterator
    of square of a number"""def __init__(self,x=0):
        self.x = x                  #x will be
assigned as 3def __iter__(self):
        self.n = 0                  #n is initialized
to 0
        return selfdef __next__(self):
        if self.n <= self.x:        #calculate square
if <=3
            result = self.n*self.n   #squaring a number
            self.n += 1              #increment number
            return result
        else:
            raise StopIteration     #if n>3, throw
exceptionfor i in square(3):
    print(i)Output:0
1
4
9
```