ITERATORS

Iteration — Iteration a general term for taking each item of something, one after another. Any time you use a loop, explicit or implicit to go over a group of items, that is iteration.

1) num = [1, 2, 3]
   for i in num:
       print (i)

Output

1
2
3

Iterator — An Iterator is an object that allows the programmer to traverse through a sequence of data without having to store the entire data in the memory.

2) l = [i for i in range (1, 1000)]
   # for i in l
       # print (i*2)

import sys
print (sys.getsizeof(l) / 64)
x = range (1, 100000000000)
print (sys.getsizeof(x) / 1024)

Output
83.1796875
0.046875

3) print (x)
O/P range (1, 100000000000)

Iterable

Iterable is an object, which one can iterate over. It generates an Iterator when passed to iter() method.

1) L = [1, 2, 3]
type(L)

O/P list

type(iter(l))

O/P list_iterator)

Point to remember

Every Iterator is also and Iterable
Not all Iterables are Iterators

Trick

Every Iterable has an iter function.
Every Iterator has both iter function as well as a next function.

2) a = 2
a
for i in a:
print(i)

dir(a)

Output
all directory

3) t = {1:2, 3:4}
dir(t)

Output

④ l = [1,2,3]

L is not an iterator

iter_L = iter(L)

iter_L is an iterator

next(iter_l)

O/P 1

next(iter_l)

O/P 2

Understanding how for loop works

```
num = [1,2,3]
for i in num:
    print(i)
```

Output

1
2
3

```
num = [1,2,3]
# fetch
iter_num = iter(num)
# step2 -> next
next(iter_num)
next(iter_num)
next(iter_um)
next(iter_num)
```

Making our own for loop

```
def mera_Khudka_for_loop(iterable):
    iterator = iter(iterable)
    while True:
        try:
            print(next(iterator))
        except StopIteration:
            break
```

```
a = [1,2,3]
b = range(1,11)
c = (1,2,3)
d = {1,2,3}
e = {0:1, 1:1}
mera_khudka_for_loop(e)
```

O/P

0
1

A confusing point

```
num = [1,2,3]
iter_obj = iter(num)
print(id(iter_obj), 'Address of iterater 1')
iter_obj2 = iter(iter_obj)
print(id(iter_obj2), 'Address of iterater 2')
```

Output

2250 Address of iterater 1
22503 " " 2

Let's create our own range() function.

```
class own_range:
    def __init__(self, start, end):
        self.start = start
        self.end = end
    def __iter__(self):
        return own_range_iterator(self)
class own_range_iterator:
    def __init__(self, iterable_obj):
        self.iterable = iterable_obj
    def __iter__(self):
        return self
    def __next__(self):
        if self.iterable.start >= self.iterable.end:
            raise StopIteration
```

```
        current = self.iterable.start
        self.iterable.start += 1
        return current

x = own_range(1,11)
for i in own_range(1,11):
    print(i)
```

Output

```
2
3
4
5
6
7
8
9
10
```

```
type(x)
```

O/P `<--main--.own_range`

```
iter(x)
```

O/P `<--main--.own_range_iterator at 0x2140208a5c10>`