Iterables and Iterators

In [1]:

```
# list is iterables

lst = [1,2,3,4,5,6,7,8,9]
for i in lst:
    print(i)

    # whenever we save a list as iterables all its value are saved in memory and get init
iallised,
    # as soon as we iterate it all values get printed.
```

```
1
2
3
4
5
6
7
8
9
```

Converting iterables into iterator.

iter() is an inbuilt function actually convert a list into an iterator.

In [2]:

```
iter(lst)

        # whenever we save a list as iterator all its value are saved in memory
        # and only 1 value gets initiallised,
        # as soon as we iterate it through next() only 1 values get printed
```

Out[2]:

```
<list_iterator at 0x54ec288>
```

In [3]:

```
lst.__iter__()    ## same as iter(lst)
```

Out[3]:

```
<list_iterator at 0x54ae7c8>
```

In [4]:

```
lst1 = iter(lst)
```

In [5]:

```
lst1
```

Out[5]:

```
<list_iterator at 0x54cfa08>
```

In [6]:

```
next(lst1)
```

Out[6]:

```
1
```

In [7]:

```
next(lst1)
```

Out[7]:

```
2
```

In [8]:

```
next(lst1)
```

Out[8]:

```
3
```

In [9]:

```
lst1 = iter(lst)
```

In [10]:

```
for i in lst1:
    print(i)
```

```
1
2
3
4
5
6
7
8
9
```

why we use iterator inplace of iterables?
Because when we deals with millions of data and use iterables our memory will get consume unneccesory so we use iterator in place of iterables

Generator

In [11]:

```
def square(n):
    for i in range(n):
        return i**2
```

In [12]:

```
square(3)
```

Out[12]:

```
0
```

In [13]:

```
## In Generator we use yeild in place of return.
def square(n):
    for i in range(n):
        yield i**2
```

In [14]:

```
square(3)
```

Out[14]:

```
<generator object square at 0x000000000551F4C8>
```

In [15]:

```
for i in square(3):
    print(i)
```

```
0
1
4
```

In [16]:

```
for i in square(2):
    print(i)
```

```
0
1
```

In [17]:

```
a = square(3)
```

In [18]:

```
next(a)
```

Out[18]:

```
0
```

In [19]:

```
next(a)
```

Out[19]:

```
1
```

In [20]:

```
print(a.__next__())     # same as next(a)
```

```
4
```

toh pata yeh chala ki :-

1.generator are similar to iterator, infact it used to create complex iterator,

1. **to create iterator we use iter() and for generator we use function along with yeild keyword.**
2. **generator uses yeild keyword, means yeild keyword saves and can return the local variable., so we can use for loops with generator.**
3. **generator in python is faster than iterator**
4. **iterator is more memory efficient.**

In [21]:

```
## example
name = "harry"
iterator1 = name.__iter__()
print(iterator1.__next__())
print(iterator1.__next__())
print(iterator1.__next__())
print(iterator1.__next__())
print(iterator1.__next__())
print(iterator1.__next__())
```

```
h
a
```

```
r
y
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-21-91d7fc130ab2> in <module>
      7 print(iterator1.__next__())
      8 print(iterator1.__next__())
----> 9 print(iterator1.__next__())

StopIteration:
```

In [22]:

```
try:
    name = "harry"
    iterator1 = name.__iter__()
    print(iterator1.__next__())
    print(iterator1.__next__())
    print(iterator1.__next__())
    print(iterator1.__next__())
    print(iterator1.__next__())
    print(iterator1.__next__())
except StopIteration:
    print("Value exceeded")
```

```
h
a
r
r
y
Value exceeded
```