**Difference Between Iterator VS Generator**

Last Updated : 15 Apr, 2024

A process that is repeated more than one time by applying the same logic is called an Iteration.  In programming languages like python, a loop is created with few conditions to perform iteration till it exceeds the limit. If the loop is executed 6 times continuously, then we could say the particular block has iterated 6 times.

**Example:**

1

a = [0, 5, 10, 15, 20]

2

for i in a:

3

if i % 2 == 0:

4

print(str(i)+' is an Even Number')

5

else:

6

print(str(i)+' is an Odd Number')

**Output:**

*0 is an Even Number*

*5 is an Odd Number*

*10 is an Even Number*

*15 is an Odd Number*

*20 is an Even Number*

**Iterator**

An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc. Iterators are implemented using a class and a local variable for iterating is not required here, It follows lazy evaluation where the evaluation of the expression will be on hold and stored in the memory until the item is called specifically which helps us to avoid repeated evaluation. As lazy evaluation is implemented, it requires only 1 memory location to process the value and when we are using a large dataset then, wastage of RAM space will be reduced the need to load the entire dataset at the same time will not be there.

Using an iterator-

* **iter()** function is used to create an iterator containing an iterable object.
* **next(**) function is used to call the next element in the iterable object.
* After the iterable object is completed, to use them again reassign them to the same object.

**Example:**

1

iter\_list = iter(['Geeks', 'For', 'Geeks'])

2

print(next(iter\_list))

3

print(next(iter\_list))

4

print(next(iter\_list))

**Output:**

*Geeks*

*For*

*Geeks*

**Generators**

It is another way of creating iterators in a simple way where it uses the keyword “yield” instead of returning it in a defined function. Generators are implemented using a function. Just as iterators, generators also follow lazy evaluation. Here, the yield function returns the data without affecting or exiting the function. It will return a sequence of data in an iterable format where we need to iterate over the sequence to use the data as they won’t store the entire sequence in the memory.

**Example:**

1

def sq\_numbers(n):

2

for i in range(1, n+1):

3

yield i\*i

4

​

5

​

6

a = sq\_numbers(3)

7

​

8

print("The square of numbers 1,2,3 are : ")

9

print(next(a))

10

print(next(a))

11

print(next(a))

**Output:**

*The square of numbers 1,2,3 are :*

*1*

*4*

*9*

**Table of difference between Iterator vs Generators**

| **Iterator** | **Generator** |
| --- | --- |
| Class is used to implement an iterator | Function is used to implement a generator. |
| Local Variables aren’t used here. | All the local variables before the yield function are stored. |
| Iterators are used mostly to iterate or convert other objects to an iterator using iter() function. | Generators are mostly used in loops to generate an iterator by returning all the values in the loop without affecting the iteration of the loop |
| Iterator uses iter() and next() functions | Generator uses yield keyword |
| Every iterator is not a generator | Every generator is an iterator |