# Generators in Python

1. **Generator-Function**

A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

# A generator function that yields 1 for first time,

# 2 second time and 3 third time

def simpleGeneratorFun():

    yield 1

    yield 2

    yield 3

# Driver code to check above generator function

for value in simpleGeneratorFun():

    print(value)

Output :

1

2

3

1. **Generator-Object**

Generator functions return a generator object. Generator objects are used either by calling the next method on the generator object or using the generator object in a “for in” loop (as shown in the above program).

# A Python program to demonstrate use of generator object with next()

# A generator function

def simpleGeneratorFun():

    yield 1

    yield 2

    yield 3

# x is a generator object

x = simpleGeneratorFun()

# Iterating over the generator object using next

print(x.next()) # In Python 3, \_\_next\_\_()

print(x.next())

print(x.next())

1. Fibonacci series using generator

# A simple generator for Fibonacci Numbers

def fib(limit):

    # Initialize first two Fibonacci Numbers

    a, b = 0, 1

    # One by one yield next Fibonacci Number

    while a < limit:

        yield a

        a, b = b, a + b

# Create a generator object

x = fib(5)

print("\nUsing for in loop")

for i in fib(5):

    print(i)

A more practical type of stream processing is handling large data files such as log files. Generators provide a space efficient method for such data processing as only parts of the file are handled at one given point in time.