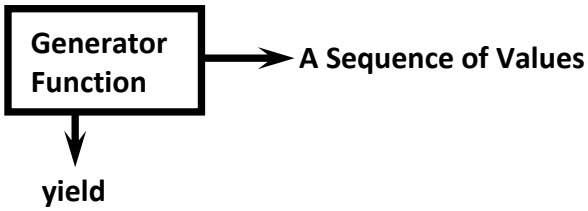
# Learn Complete Python In Simple Way

# GENERATOR FUNCTIONS

## STUDY MATERIAL

Generator is a function which is responsible to generate a sequence of values.
We can write generator functions just like ordinary functions, but it uses yield keyword to return values.




```
1)  def mygen():
2)      yield 'A'
3)      yield 'B'
4)      yield 'C'
5)
6)  g=mygen()
7)  print(type(g))
8)
9)  print(next(g))
10) print(next(g))
11) print(next(g))
12) print(next(g))
```

**Output**

```
<class 'generator'>
A
B
C
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print(next(g))
StopIteration
```

```
1)  def countdown(num):
2)      print("Start Countdown")
3)      while(num>0):
4)          yield num
5)          num=num-1
6)  values=countdown(5)
7)  for x in values:
8)      print(x)
```



**Output**

```
Start Countdown
5
4
3
2
1
```

**Eg 3:** To generate first n numbers

```
1)  def firstn(num):
2)      n=1
3)      while n<=num:
4)          yield n
5)          n=n+1
6)
7)  values=firstn(5)
8)  for x in values:
9)      print(x)
```

**Output**

```
1
2
3
4
5
```

We can convert generator into list as follows:

```
values = firstn(10)
l1 = list(values)
print(l1)    #[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

**Eg 4:** To generate Fibonacci Numbers...
The next is the sum of previous 2 numbers

**Eg:** 0,1,1,2,3,5,8,13,21,...

```
1)  def fib():
2)      a,b=0,1
3)      while True:
4)          yield a
5)          a,b=b,a+b
6)  for f in fib():
```

```
7)    if f>100:
8)        break
9)    print(f)
```

**Output**

```
0
1
1
2
3
5
8
13
21
34
55
89
```

## Advantages of Generator Functions:

1) When compared with Class Level Iterators, Generators are very easy to use.
2) Improves Memory Utilization and Performance.
3) Generators are best suitable for reading Data from Large Number of Large Files.
4) Generators work great for web scraping and crawling.

## Generators vs Normal Collections wrt Performance:

```
1)  import random
2)  import time
3)
4)  names = ['Sunny','Bunny','Chinny','Vinny']
5)  subjects = ['Python','Java','Blockchain']
6)
7)  def people_list(num_people):
8)      results = []
9)      for i in range(num_people):
10)         person = {
11)                 'id':i,
12)                 'name': random.choice(names),
13)                 'subject':random.choice(subjects)
14)                }
15)         results.append(person)
16)     return results
```

```
17)
18) def people_generator(num_people):
19)     for i in range(num_people):
20)         person = {
21)                   'id':i,
22)                   'name': random.choice(names),
23)                   'major':random.choice(subjects)
24)                  }
25)         yield person
26)
27) """t1 = time.clock()
28) people = people_list(10000000)
29) t2 = time.clock()"""
30)
31) t1 = time.clock()
32) people = people_generator(10000000)
33) t2 = time.clock()
34)
35) print('Took {}'.format(t2-t1))
```

**Note:** In the above program observe the differnce wrt execution time by using list and generators

## Generators vs Normal Collections wrt Memory Utilization:

### Normal Collection:

```
l=[x*x for x in range(10000000000000000)]
print(l[0])
```

We will get MemoryError in this case because all these values are required to store in the memory.

### Generators:

```
g=(x*x for x in range(10000000000000000))
print(next(g))
```

**Output:** 0

We won't get any MemoryError because the values won't be stored at the beginning