Generator function or generators
Generators are used for creating iterator objects
Generator functions allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop.
Generators in Python are used to create iterators and return a traversal object. It helps in traversing all the items one at a time with the help of the keyword yield.
Normal function returns value using return keyword.
Generator function returns value using yield keyword.

What is difference between return and yield?

return	**yield**
This keyword return value, after returning value it terminates execution of function.	Yield keyword return value, after returning value, it pause execution of function. when iterated it resume back and continue where it is paused.
Function with return keyword is not generator function	Function with yield is called generator function, which return iterator object.

Example

```
def fun1():
    yield 1
    yield 2
    yield 3
    yield 4
    yield 5

def fun2():
    return 1
    return 2
    return 3
    return 4
    return 5



f1=fun1() # Creating iterator
```

Output

```
<generator object fun1 at
0x0000024DCAB70040>
1 2 3
4 5
1
```

```
object
print(f1)
value1=next(f1)
value2=next(f1)
value3=next(f1)
print(value1,value2,value3)
for value in f1:
    print(value,end=' ')

print()
f2=fun2()
print(f2)
```

Example:

```
def sqr_generator(m,n):
    for num in range(m,n+1):
       yield num**2



generator1=sqr_generator(1,5)
print(generator1)
for value in generator1:
    print(value)

r1=range(1,6)
for value in r1:
    print(value)
```

Output

```
<generator object
sqr_generator at
0x000002610475F3E0>
1
4
9
16
25
1
2
3
4
5
```

Example

```
def rev_iter(seq):
    for value in seq[::-1]:
        yield value

list1=[10,20,30,40,50,60,70,80,90,
100]
a=iter(list1)
for value in a:
    print(value)

b=rev_iter(list1)
```

Output

```
10
20
30
40
50
60
70
80
90
100
100
```

```
for value in b:
    print(value)
```

```
90
80
70
```

Example

```
def
float_range(start,stop,step=1.0):
    if start<stop:
        while start<stop:
            yield start
            start=start+step
    elif start>stop:
        while start>stop:
            yield start
            start=start+step

a=range(1,6)
for value in a:
    print(value)

b=float_range(1.0,6.0)
for value in b:
    print(value,end=' ')

print()
c=float_range(6.0,0.0,-1)
for value in c:
    print(value,end=' ')
```

Output

```
1
2
3
4
5
1.0 2.0 3.0 4.0 5.0
6.0 5.0 4.0 3.0 2.0 1.0
```

Generator expression
Generator expression is a single line statement, which return generator iterator object.

Syntax:
<variable-name>=(expression for variable in iterable if test)
Note: this is similar to comprehensions.

Example:	**Output**

```
alpha_generator=(chr(n) for n in range(65,91))
for value in alpha_generator:
    print(value,end=' ')

print()
even_generator=(n for n in range(1,21) if n%2==0)
for value in even_generator:
    print(value,end=' ')

print()
odd_generator=(n for n in range(1,21) if n%2!=0)
for value in odd_generator:
    print(value,end=' ')
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19
```

What is difference between generator function and generator expression?

A generator is a type of iterable in Python that allows you to iterate over a sequence of values one at a time. It is defined using the *yield* keyword, and the generator function is called like any other function. However, instead of returning a value, the generator function yields a sequence of values, one at a time, when it is iterated over. This allows you to create large sequences of values without using up a lot of memory.

A generator expression is a concise way to create a generator object. It is similar to a list comprehension, but it returns a generator instead of a list. Generator expressions are defined using parentheses, and they use the same syntax as list comprehensions, but with a single element on the right-hand side of the expression.

Lambda functions or lambda expressions or Anonymous functions