Python provides a generator to create your own iterator function.

generator is a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values. In a generator function, a yield statement is used rather than a return statement.

```
def mygenerator():
    print('First item')
    yield 10

    print('Second item')
    yield 20

    print('Last item')
    yield 30

gen = mygenerator()

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

The generator function cannot include the return keyword. If you include it, then it will terminate the function. The difference between yield and return is that yield returns a value and pauses the execution while maintaining the internal states, whereas the return statement returns a value and terminates the execution of the function.

Using for Loop with Generator Function

The generator function can also use the for loop.

```
def get_sequence_upto(x):
    for i in range(x):
        yield i


a=get_sequence_upto(10)

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

The generator is called just like a normal function. However, its execution is paused on encountering the yield keyword. This sends the first value of the iterator stream to the calling environment. However, local variables and their states are saved internally.

Note:

One of the advantages of the generator over the iterator is that elements are generated dynamically. Since the next item is generated only after the first is consumed, it is more memory efficient than the iterator.

Generator Expression

Python also provides a generator expression, which is a shorter way of defining simple generator functions. The generator expression is an anonymous generator function.

```
squres = (x*x for x in range(5))

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

`(x*x for x in range(5))` is a generator expression.

The first part of an expression is the `yield` value and the second part is the for loop with the collection.

The generator expression can also be passed in a function. It should be passed without parentheses, as shown below.

```
import math

ans=sum(x*x for x in range(5))
print(ans)
```