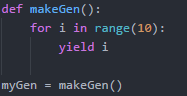
**Generators:**

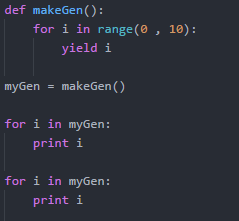
In Python generators are iterable just like lists. But the major difference between these two iterables is that in case of lists we can iterate over them as many times as we want, but in case of generators we can only iterate over them for once. That is, it can be said that it is a “onetime use” iterable. This is because no memory is allocated when generators are created, they are created on the fly. Generators are used whenever we need to iterate over some values for only once in the whole program. There are more than one way to initialize generators.



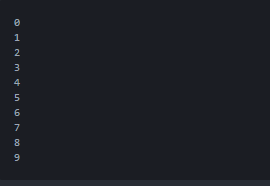
Here the yield keyword is responsible for the creation of the generator. The yield keyword acts almost like the return keyword but there are differences. The yield keyword only returns generator. It doesn’t end the function like return.

Things are going to get a bit tricky now: whenever the function makeGen() is called here, the program doesn’t run the whole function body but only returns a generator object due to the keyword yield. The function is only run each time we try to iterate over the generator until it hit the keyword yield. With return the local variables are all destroyed until that function is called again, but with yield the function doesn’t halt. The function continues to return the “yielded” value until there is nothing left to yield.

Enough theories!! Now lets iterate over the generator:

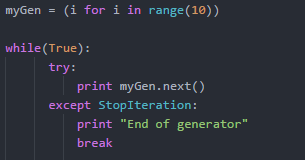


The output of the above written code is:



Note that the values from 0 to 9 are only printed once although in our code we have two for loops both asking to iterate over the generator. But like I said before, a generator is a “onetime use” iterable. We cannot iterate over the generator for more than once.

There is another way to initialize generators:



The next() method of a generator object is used as it iterator and points to the next value of the generator starting from null. It will also raise StopIteration when generator has reached its end.