Iterators & Generators

Generator ~~fun~~ functions allow us to write a function that can send back a value & then later resume to pick up where it left off.
It allows us to generate a sequence of values over time. The main diff. is use of yield statement.

→ The main-diff. is when a generator funcⁿ is compiled they become an object that supports iteration.
→ They will automatically suspend & resume their execution
→ They automatically suspend & resume their execution & state around the last point of value generation.
→ Main-advantage:-
Instead of computing entire series of values up front, the generator computes one value & suspends it's activity waiting for next instruction.
→ This is called state suspension

E.g.

```
def gencubes(n):
    for num in range(n):
        yield num**3
```

```
list(gencubes(6))
```

O/p: [0, 1, 8, 27, 64, 125]

Eg. def genfibon(n):

```
        a=1
        b=1
        for in in range(n):
            yield a
            a,b=b,a+b
```

~~list~~(genfibon(5))

O/P 1 1 2 3 5

next() → Access the next term in the sequence

In: g = genfibon()

In: ~~next~~ print(next(g))
O/p: 1

In: print (next (g))
O/p: 1

In: print(next (g))
O/p: 2

String object supports iteration but isn't a iterator.

Iter() → It allows us to iterate over a non-iterable object

```
In: s = 'hello'
In: s_it = iter(s)

In: next(s_it)
o/p: 'h'

In: next(s_it)
o/p: 'e'
```