Iterator :-

- iterator is an object, which impliments iterator protocall
- which consist of **iter**() and **next**() function
- for loop iself is an example of iterator
  - iterator - we can traverse through all the values present in a sequence or itarable
  - iterable - it is like a sequence such as list, str, tuple, etc...
- it has two functions
  - iter function
  - next function
- "_" - temperory use

we can also create our own iterators:-

In [4]:
```
class Range():
    def __init__(self, start, last):
        self.start = start
        self.last = last

    def __iter__(self):
        self.start

    def __next__(self):
        if self.start < self.last:
            sqre = self.start**2
            print(sqre)
            self.start += 1

ran = Range(1,10)

ran.__iter__()
ran.__next__()
ran.__iter__()
ran.__next__()
```

```
1
4
```

In [ ]:
```

```

Generators :-

- Generator-Function: A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the "yield" keyword rather than return.
- If the body of a def contains yield, the function automatically becomes a generator function.
- They process the data incrementally and do not allocate memory to all the results at the same time.

where do we use genetators

- Consider using Generator when dealing with a huge dataset.
- Consider using Generator in scenarios where we do NOT need to reiterate it more than once.
- Generators give us lazy evaluation.
- They are a great way to generate sequences in a memory-efficient manner.

In [7]:

```python
def square():
    for i in range(1,6):
        yield i

sqr = square()
```

In [9]:

```python
print(next(sqr))
print(next(sqr))
```

```
2
3
```

In [ ]:

```

```