Iterators & Generators

Besant Technologies

Iterators:

- Objects that can be iterated.
- Ex: for loop
- Two methods:
  - __iter__()
  - __next__()

* Can create an iterator from sequences

* iter() - calls __iter__() method and returns an iterator object

Ex:

- my_list = [4, 7, 0, 3]
- my_iter = iter(my_list)
- print(next(my_iter))
- print(my_iter.__next__())
- next() will throw error at the end of sequence.

Generators:

- Function which returns iterators.
- Yield instead of return.
- Normal function with at least one yield - Generators.
- Yield: Passes the function execution, saves all states of variables and continues from there in successive calls.
- Can use next() on generator functions to iterate.
- Can use for loop

Generator expressions:

- Same as list comprehensions but with ‘()’ brackets.
- Ex: (x**2 for x in my_list)

any() and all()

- any() - returns false if all elements of a sequence is false
- all() - returns true if all the elements of a sequence is true.
- False - 0, False, empty sequence
- all() - empty sequence also gives True.