# Iterators

Iterator concept is introduced in ES6 and it is used to loop over set of values/collection.

By definition iterator must implement **next()** function : return object {value, done}, where value is next value from collection in iteration sequence and done is true/false mentioning if sequence/connection is finished.

In JavaScript we can loop through arrays of different objects using for loop but what if we have some complex object which is not supported by for loop? what if we want to return custom object by loping through complex object? We can use iterators in this case.

Lets understand this with very simple object of list/array rather than directly going to complex object. Even though list/array can be looped through for loop lets see how we can implement our iterator on them.

function citiIterator(cities){

    let index = 0;

    // creating return as object with property 'next'. This property

    // holds function which then returns next value from provided collection

    // and increment the counter

    let ret\_val = {

        next : function(){

                // check if we have still not reached end of collection then return

                // next value or return done=false object

                return index < cities.length ?

                     { done : false, vaue : cities[index++]}

                     : {done : true}

                }

            }

    return ret\_val;

}

let citiList = ['Pune','Mumbai','Hyderabad','Bangalore'];

let myIterator = citiIterator(citiList);

console.log(myIterator.next());

console.log(myIterator.next());

//==> Pune,Mumbai

# Generator

Used to generate and yield next values. Yield automatically returns {value : <<>>, done:<<>>} syntax.

function\* gen() {

    yield 1;

    yield 2;

    yield 3;

  }

  const g = gen();

  console.log(g.next()); // { value: 1, done: false }

  console.log(g.next()); // { value: 2, done: false }