Iterator and Generator

## Generator

A Generator Function is a special type of function that works as a factory for iterators. When it is executed it returns a new Generator object. A function becomes a Generator Function if it uses the [**function\***](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function*) syntax.

A generator function can return values one at a time by pausing execution at each iteration. When you create an instance of a generator, these items can be accessed using an iterator. This is the general syntax for creating a generator function:

function \*genFunc () {

    yield value;

}

The **\*** signifies that this is a generator function. The **yield** keyword pauses our function and supplies the state of the generator at that particular moment.

Example

function\* myValue (a)

{

let index = 0;

while (index <= a)

{

console.log (index)

yield index++;

}

}

myValue (200).next ();

## Iterator

An object is an **iterator** when it knows how to access items from a collection one at a time, while keeping track of its current position within that sequence. In JavaScript an iterator is an object that provides a next () method which returns the next item in the sequence. This method returns an object with two properties: done and value.

**Iterator** is an object that is returned by invoking [Symbol.iterator]() on an iterable. Using its next() method, it wraps around each iterated element in the data structure and returns it one by one.

**IteratorResult**: A new data structure returned by next()

Example

var myAr = [2,4,55,11,10,23,56,87];

function iterator(name)

{

let index = 0;

return {

next :function()

{

return index < name.length ? { value : name[index++],done:false}:{done:true}

}

}

}

console.log( iterator(myAr).next().value);