ES6 introduces a new mechanism for traversing data: iteration. Two concepts are central to iteration:

* An iterable is a data structure that wants to make its elements accessible to the public. It does so by implementing a method whose key is Symbol.iterator. That method is a factory for iterators.
* An iterator is a pointer for traversing the elements of a data structure (think cursors in databases).

Expressed as interfaces in TypeScript notation, these roles look like this:

**interface** Iterable {

[Symbol.iterator]() : Iterator;

}

**interface** Iterator {

next() : IteratorResult;

}

**interface** IteratorResult {

value: any;

done: **boolean**;

}

The following values are iterable:

• Arrays

• Strings

• Maps  
• Sets

• DOM data structures (work in progress)  
*Plain objects are not iterable*

Iterability

* Data consumers: JavaScript has language constructs that consume data. For example, for- of loops over values and the spread operator (...) inserts values into Arrays or function calls.
* Data sources: The data consumers could get their values from a variety of sources. For example, you may want to iterate over the elements of an Array, the key-value entries in a Map or the characters of a string.

So data consumers use iterable data sources to perform some action

Data consumers have a mechanism to iterate over iterable elements – this is known as an iterator

This relationship is centered around a new ES6 interface known as Iterable

Data consumers tap into the iterables that data sources implement

They are the mechanism which enable iteration

• Source: A value is considered iterable if it has a method whose key is the symbol Symbol.iterator that returns a so-called iterator. The iterator is an object that returns values via its method next(). We say: it iterates over the items (the content) of the iterable, one per method call.

• Consumption: Data consumers use the iterator to retrieve the values they are consuming.