The Role of the Iterable Interface

understand iterators and their data flow, create independent iterators and connect the dots using ES6

We can understand iterables a bit better by concentrating on data flow:

- The `for-of` loop, the Spread operator, and some other language constructs are *data consumers*. They consume iterable data.
- Iterable data structures such as arrays, strings, dom data structures, maps, and sets are *data sources*
- The *iterable interface* specifies how to connect data consumers with data sources
- *Iterable objects* are created according to the iterable interface specification. Iterable objects can create iterator objects that facilitate the iteration on their data source, and prepare the result for a data consumer.

We can create independent iterator objects on the same iterable. Each iterator acts like a pointer to the upcoming element the linked data source can consume.

In the lesson on sets and maps, you have learned that you can convert sets to arrays using the spread operator:

```
let arr = [...set];
```

You now know that a set is an iterable object, and the spread operator is a data consumer. The formation of the array is based on the iterable interface. ES6 makes a lot of sense once you start connecting the dots. Now, let's talk about generators in the next lesson.