Array Iterator Methods

[(https://github.com/learn-co-curriculum/phase-1-array-iterator-methods)](https://github.com/learn-co-curriculum/phase-1-array-iterator-methods) [(https://github.com/learn-co-curriculum/phase-1-array-iterator-methods/issues/new)](https://github.com/learn-co-curriculum/phase-1-array-iterator-methods/issues/new)

Learning Goals

- Understand how JavaScript's iterator methods help us

Introduction

We've learned about `Array` methods that allow us to modify arrays by adding and removing elements (`push()` , `slice()` , etc). These methods operate on the array as a whole, but JavaScript also includes methods that assist us in iterating through an array and interacting with each individual element in some way. We will be learning about these methods in the next few lessons.

Why Use JavaScript's Iterator Methods?

Imagine that we have a collection of Flatbook user objects in an array:

```
const users = [
  {
    firstName: "Niky",
    lastName: "Morgan",
    favoriteColor: "Blue",
    favoriteAnimal: "Jaguar",
    personalQuote: "You're never too old to learn something new",
  },
  {
    firstName: "Tracy",
    lastName: "Lum",
    favoriteColor: "Yellow",
    favoriteAnimal: "Penguin",
    personalQuote: "I just got lost in thought - it was unfamiliar territory",
  },
  {
    firstName: "Josh",
    lastName: "Rowley",
    favoriteColor: "Blue",
    favoriteAnimal: "Penguin",
    personalQuote: "Always remember you’re unique, just like everyone else",
  },
  {
    firstName: "Kate",
    lastName: "Travers",
    favoriteColor: "Red",
    favoriteAnimal: "Jaguar",
    personalQuote: "Behind every great man is a woman rolling her eyes",
  },
  {
    firstName: "Avidor",
    lastName: "Turkewitz",
    favoriteColor: "Blue",
    favoriteAnimal: "Penguin",
    personalQuote:
      "You don’t have to see the whole staircase, just take the first step",
  },
  {
    firstName: "Drew",
    lastName: "Price",
    favoriteColor: "Yellow",
    favoriteAnimal: "Elephant",
    personalQuote:
      "Failure is not the opposite of success: it’s part of success",
  },
];
```

We can iterate over that collection and print out everyone's first name as follows:

```
function firstNamePrinter(collection) {
  for (const user of collection) {
    console.log(user.firstName);
  }
}

firstNamePrinter(users);
// LOG: Niky
// LOG: Tracy
// LOG: Josh
// LOG: Kate
// LOG: Avidor
// LOG: Drew
```

It's also not too difficult to print out only users whose favorite color is blue:

```
function blueFilter(collection) {
  for (const user of collection) {
    if (user.favoriteColor === "Blue") {
      console.log(user.firstName);
    }
  }
}

blueFilter(users);
// LOG: Niky
// LOG: Josh
// LOG: Avidor
```

But what if we wanted to print out users whose favorite color is red instead? With the above approach, we'd need to create a whole new `redFilter` function.

We can improve matters by abstracting out the color into a variable:

```
function colorFilter(collection, color) {
  for (const user of collection) {
    if (user.favoriteColor === color) {
      console.log(user.firstName);
    }
  }
}

colorFilter(users, "Red");
// LOG: Kate
```

This is definitely better: now we can print a list of users with any favorite color, but what if we want to print out users with a particular favorite animal instead? Or what if we want to do something other than simply print a list to the screen? For example, we might want to access each user's personal quote and add an exclamation point to the end. Or, using a different example, we might want to calculate the average amount of money spent by our customers in a given month.

There are any number of ways we might want to interact with the elements in our array, but all of them have some things in common: in all cases, we 1) pass in an array, 2) iterate through it using some sort of loop, and 3) write some code to interact with each element in the desired way.

Accessing and interacting with the elements in an array is, in fact, a very common need in programming. So common that JavaScript has created a number of built in methods that reduce our need to write the common parts of the process over and over again. Below is a summary of some common use cases and the JavaScript method(s) we can use for each:

Use Case	Method
Finding a single element that meets a condition	`indexOf()` , `find()`
Finding and returning a list of elements that meet a condition	`filter()`
Modifying each element and returning the modified array	`map()`
Creating a summary or aggregation of values in an array	`reduce()`

We will learn about each of the above methods in the upcoming lessons.

Resources

- MDN
  - **Array (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)**