Iterators

This chapter explores the functional programming side of JS previously seen in Scala.
Specifically, higher-order functions`

Functions are first class objects, with properties .length, .name and method .toString()

Functions Assigned to Variables

In JavaScript, functions are a data type just as strings, numbers, and arrays are data types. Therefore, functions can be assigned as values to variables, but are different from all other data types because they can be invoked.

```
let plusFive = (number) => {
  return number + 5;
};
// f is assigned the value of plusFive
let f = plusFive;

plusFive(3); // 8
// Since f has a function value, it can
be invoked.
f(9); // 14
```

Callback Functions

In JavaScript, a callback function is a function that is passed into another function as an argument. This function can then be invoked during the execution of that higher order function (that it is an argument of). Since, in JavaScript, functions are objects, functions can be passed as arguments.

To put in the callback function, you write the function name, or the anonymous function definition

```
const isEven = (n) => {
  return n % 2 == 0;
}

let printMsg = (evenFunc, num) => {
  const isNumEven = evenFunc(num);
  console.log(`The number ${num} is an
even number: ${isNumEven}.`)
}

// Pass in isEven as the callback
function
printMsg(isEven, 4);
// Prints: The number 4 is an even
number: True.
```

Higher-Order Functions

In Javascript, functions can be assigned to variables in the same way that strings or arrays can. They can be passed into other functions as parameters or returned from them as well.
A “higher-order function” is a function that accepts functions as parameters and/or returns a function.

Array Method `.reduce()` aka FOLD!!

The `.reduce()` method iterates through an array and returns a single value.
It takes a callback function with two parameters `(accumulator, currentValue)` as arguments. On each iteration, `accumulator` is the value returned by the last iteration, and the `currentValue` is the current element. Optionally, a second argument can be passed which acts as the initial value of the accumulator.
Here, the `.reduce()` method will sum all the elements of the array.

```
const arrayOfNumbers = [1, 2, 3, 4];

const sum =
arrayOfNumbers.reduce((accumulator,
currentValue) => {
  return accumulator + currentValue;
});

console.log(sum); // 10
```

Array Method `.forEach()`

The `.forEach()` method executes a callback function on each of the elements in an array in order.
Here, the callback function containing a `console.log()` method will be executed `5` times, once for each element.

```
const numbers = [28, 77, 45, 99, 27];

numbers.forEach(number => {
  console.log(number);
});
```

Array Method `.filter()`

The `.filter()` method executes a callback function on each element in an array. The callback function for each of the elements must return either `true` or `false`. The returned array is a new array with any elements for which the callback function returns `true`.
Here, the array `filteredArray` will contain all the elements of `randomNumbers` but `4`.

```
const randomNumbers = [4, 11, 42, 14,
39];
const filteredArray =
randomNumbers.filter(n => {
  return n > 5;
});
```

Array Method `.map()`

The `.map()` method executes a callback function on each element in an array. It returns a new array made up of the return values from the callback function.
The original array does not get altered, and the returned array may contain different elements than the original array.

```
const finalParticipants = ['Taylor',
'Donald', 'Don', 'Natasha', 'Bobby'];

const announcements =
finalParticipants.map(member => {
  return member + ' joined the
contest.';
})

console.log(announcements);
```

forEach vs. map

1. The callback function for forEach() has to return undefined.
The callback function for map() can return values, which are then the values replacing the original elements (like reduce())

2. forEach() doesn't return an array, map() returns the mutated array and leaves the original array unchanged (like filter())