Higher Order Functions

In this lesson, we will go over the concept of High-Order functions and see their uses in JavaScript.

Throughout this chapter, we have leveraged the fact that JavaScript functions can be passed around just like any other value. We say that functions are *first-class citizens* in JavaScript, which means that they are treated equal to other types.

Thanks to their first-class citizenry, functions can be combined together, rendering programs even more expressive and enabling a truly functional programming style. A function that takes another function as a parameter or returns another function is called a *higher-order function*. Check out this final version of our example program.

```
const titles = movies => movies.map(movie => movie.title);
const byNolan = movie => movie.director === "Christopher Nolan";
const filter = (movies, func) => movies.filter(func);
const goodRating = movie => movie.imdbRating >= 7.5;
const ratings = movies => movies.map(movie => movie.imdbRating);
const average = array => array.reduce((sum, value) => sum + value, 0) / array.length;

console.log(titles(movieList));
const nolanMovieList = filter(movieList, byNolan);
console.log(nolanMovieList.length);
console.log(titles(filter(movieList, goodRating)));
console.log(average(ratings(nolanMovieList)));
```

We have defined helper functions that we combine to achieve the desired behaviour. The code is concise and self-describing. Since it takes the filtering function as a parameter, our own `filter()` function is an example of an higher-order function.