Pure Functions

Merely introducing some functions in a program is not enough to follow the functional programming paradigm. Whenever possible, we also need to use pure functions. This lesson will explain these functions in detail.

A *pure function* is a function that has the following characteristics:

- Its outputs depend solely on its inputs
- It has no side effect

A *side effect* is a change in program state or an interaction with the outside world. A database access or a `console.log()` statement are examples of side effects.

Given the same data, a pure function will always produce the same result. By design, a pure function is independent from the program state and must not access it. Such a function must accept *parameters* in order to do something useful. The only way for a function without parameters to be pure is to return a constant value.

Pure functions are easier to understand, combine together, and debug: contrary to their impure counterparts, there's no need to look outside the function body to reason about it. Still, a number of side effects are necessary in any program, like showing output to the user or updating a database. In functional programming, the name of the game is to create those side effects only in some dedicated and clearly identified parts of the program. The rest of the code should be written as pure functions. Let's refactor our example code to introduce pure functions.

```
// Get movie titles
const titles = movies => {
  const titles = [];
  for (const movie of movies) {
    titles.push(movie.title);
  }
  return titles;
};
```

```
// Get movies by Christopher Nolan
const nolanMovies = movies => {
  const nolanMovies = [];

  for (const movie of movies) {
    if (movie.director === "Christopher Nolan") {
      nolanMovies.push(movie);
    }
  }
  return nolanMovies;
};

// Get titles of movies with an IMDB rating greater or equal to 7.5
const bestTitles = movies => {
  const bestTitles = [];
  for (const movie of movies) {
    if (movie.imdbRating >= 7.5) {
      bestTitles.push(movie.title);
    }
  }
  return bestTitles;
};

// Compute average rating of a movie list
const averageRating = movies => {
  let ratingSum = 0;
  for (const movie of movies) {
    ratingSum += movie.imdbRating;
  }
  return ratingSum / movies.length;
};

console.log(titles(movieList));
const nolanMovieList = nolanMovies(movieList);
console.log(nolanMovieList.length);
console.log(bestTitles(movieList));
console.log(averageRating(nolanMovieList));
```

Since we only do refactoring, the program output is still the same. The program state (`movieList` and `nolanMovieList`) hasn't changed. However, all our functions are now pure; instead of accessing the state, they use parameters to achieve their desired behavior. As an added benefit, the function `averageRating()` can now compute the average rating of any movie list; it has become more *generic*.