JavaScript Functional Programming Review

Pure vs Impure Functions

- A pure function is one that always produces the same output for the same input and doesn't have any side effects. Its output depends only on its input, and it doesn't modify any external state.
- Impure functions have side effects, which are changes to the state of the program that are observable outside the function.

Functional programming

- Functional Programming is an approach to software development that emphasizes the use of functions to solve problems, focusing on what needs to be done rather than how to do it.
- Functional programming encourages the use of techniques that help avoid side effects, such as using immutable data structures and higher-order functions.
- When used correctly, functional programming principles lead to cleaner and more maintainable code

Currying

- Currying is a functional programming technique that transforms a function with multiple arguments into a sequence of functions, each taking a single argument.

Here is an example of a regular function vs a curried function:

```
// Regular function


function average(a, b, c) {

  return (a + b + c) / 3;

}


// Curried function
```

```
function curriedAverage(a) {
  return function(b) {
    return function(c) {
      return (a + b + c) / 3;
    };
  };
}

// Usage of curried function

const avg = curriedAverage(2)(3)(4);
```

- Currying can be particularly powerful when working with functions that take many arguments.
- Currying makes your code more flexible and easier to reuse.
- You can use arrow functions to create curried functions more concisely:

```
const curriedAverage = a => b => c => (a + b + c) / 3;
```

- While currying can lead to more flexible and reusable code, it can also make code harder to read if overused.