Introduction to Functional Programming

Learn what functional programming means and what it looks like in JavaScript. Learn how to discuss its merits and drawbacks.

This Section

This section of three lessons is meant to serve as a primer for the next two sections. The goal is to teach theory, while the next two sections will deal with how to apply what you learn to objects and arrays.

Introduction

Functional programming is a common paradigm in JavaScript. When we say it's a paradigm, we mean it's a way of thinking about and writing code that follows certain design principles.

Object-oriented programming aims to keep data and behavior together into a single item. Data and the methods used to work with that data should be together in one unit. Directly manipulating that data is acceptable.

Functional programming aims to separate data from behavior. It aims to keep data separate from the functions that work on it. In doing so, it results in functions that are pure and reusable.

Pure Functions

A function is said to be pure if it meets the following criteria:

- Does not affect anything outside its scope
- Does not rely on anything outside its scope
- Produces the same result for the same input every time

Here are some examples of functions that are NOT pure.

- This one uses a variable outside its scope.

```
const x = 10;

function add(y) {
    return x + y;
}
```

- This one mutates its input and in doing so, mutates an item present in its outer scope. After the function completes execution, `arr` will be modified and that change will persist outside the function.

```
function push10(arr) {
    arr.push(10);
}
```

- This one produces different results almost every time.

```
function random() {
    return Math.random() * 10;
}
```

`Array.push`, `pop`, and `splice` are impure, as they mutate the array they're working on, and therefore they do not follow functional programming.

Immutability

A big piece of functional programming is the concept of immutability. This means data should be constant and unchangeable.

In favor of `Array.push`, functional programming would have us write a function that *copies* the original array while adding the new element to it. This would keep the original array constant, untouched, and would generate a brand new array that has the extra value.

Functional programming states that even variables should not be changed. In a function, once a variable is given a value, it should keep that value for the rest of its life.

But

But...

You're probably thinking that you wouldn't be able to write anything useful if limited to pure functions. That's correct. JavaScript is useful precisely because of its ability to work with and manipulate different pieces of data.

Adding data to a database, writing to a file, and interacting with a user interface are all tasks that are impossible within a strict functional programming paradigm.

The goal of functional programming is to enable us to perform these types of tasks while using the ideas we've discussed as much as we can.

Why?

The purpose of functional programming is to allow us to manipulate data in increasingly complex ways. Using functional programming, it becomes much easier to write, reason about, and debug code as compared to object-oriented programming.

OOP will have methods that change an object's internal state. Data is being overwritten and the object is mutable. This makes debugging fairly difficult - a variable's value depends on where we are in the function and how much time has passed.

Small, pure functions that work on a small local scope make code easier to reason about and easier to debug. Keeping data constant means that code is much easier to follow, as we can be sure that a variable will have the same value everywhere we want to access it. Having an object or an array stay constant makes it easier to use, test, and perform operations with.

Future Sections

We mentioned that variables should be static and constant. If this is the case, how in the world would we write for-loops or while-loops? How would we iterate through anything without changing the value of our iterator?

The next two sections solve this problem. Learning array and object methods will enable us to eliminate over 90% of loops in our code. We replace these loops with pure functions that don't mutate their array and instead provide us

loops with pure functions that don't mutate their array and instead provide us with a new one.

These are powerful methods and once you understand them, you'll watch your code become more elegant, easier to write, and easier to think about. Let's continue learning about functional programming in the next lesson.