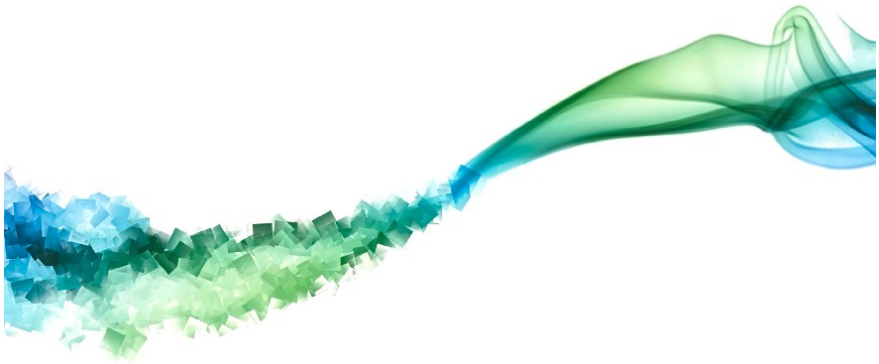
Eric Elliott
Make some magic. #JavaScript
Feb 19, 2017 · 6 min read

The Rise and Fall and Rise of Functional Programming (Composing Software)



Smoke Art Cubes to Smoke — MattysFlicks — (CC BY 2.0)

> *Note: This is part 1 of the "Composing Software" series on learning functional programming and compositional software techniques in JavaScript ES6+ from the ground up. Stay tuned. There's a lot more of this to come!*
> << Start over at Part 1 | Next >

When I was about 6 years old, I spent a lot of time playing computer games with my best friend. His family had a room full of computers. To me, they were irresistible. Magic. I spent many hours exploring all the games. One day I asked my friend, "how do we make a game?"

He didn't know, so we asked his dad, who reached up on a high shelf and pulled down a book of games written in Basic. So began my journey with programming. By the time public school got around to teaching algebra, I already knew the topic well, because programming is basically algebra. It can be, anyway.

The Rise of Composable Software

In the beginning of computer science, before most of computer science was actually done on computers, there lived two great computer scientists: Alonzo Church, and Alan Turing. They produced two

different, but equivalent universal models of computation. Both models could compute anything that can be computed (hence, “universal”).

Alonzo Church invented lambda calculus. Lambda calculus is a universal model of computation based on function application. Alan Turing is known for the turing machine. A turing machine is a universal model of computation that defines a theoretical device that manipulates symbols on a strip of tape.

Together, they collaborated to show that lambda calculus and the turing machine are functionally equivalent.

Lambda calculus is all about function composition. Thinking in terms of function composition is a remarkably expressive and eloquent way to compose software. In this text, we’re going to discuss the importance of function composition in software design.

There are three important points that make lambda calculus special:

1. Functions are always anonymous. In JavaScript, the right side of `const sum = (x, y) => x + y` is the *anonymous* function expression `(x, y) => x + y`.
2. Functions in lambda calculus only accept a single input. They’re unary. If you need more than one parameter, the function will take one input and return a new function that takes the next, and so on. The n-ary function `(x, y) => x + y` can be expressed as a unary function like: `x => y => x + y`. This transformation from an n-ary function to a unary function is known as currying.
3. Functions are first-class, meaning that functions can be used as inputs to other functions, and functions can return functions.

Together, these features form a simple, yet expressive vocabulary for composing software using functions as the primary building block. In JavaScript, anonymous functions and curried functions are optional features. While JavaScript supports important features of lambda calculus, it does not enforce them.

The classic function composition takes the output from one function and uses it as the input for another function. For example, the composition:

```
f . g
```

Can be written as:

```
compose2 = f => g => x => f(g(x))
```

Here's how you'd use it:

```
double = n => n * 2
inc = n => n + 1
```

```
compose2(double)(inc)(3)
```

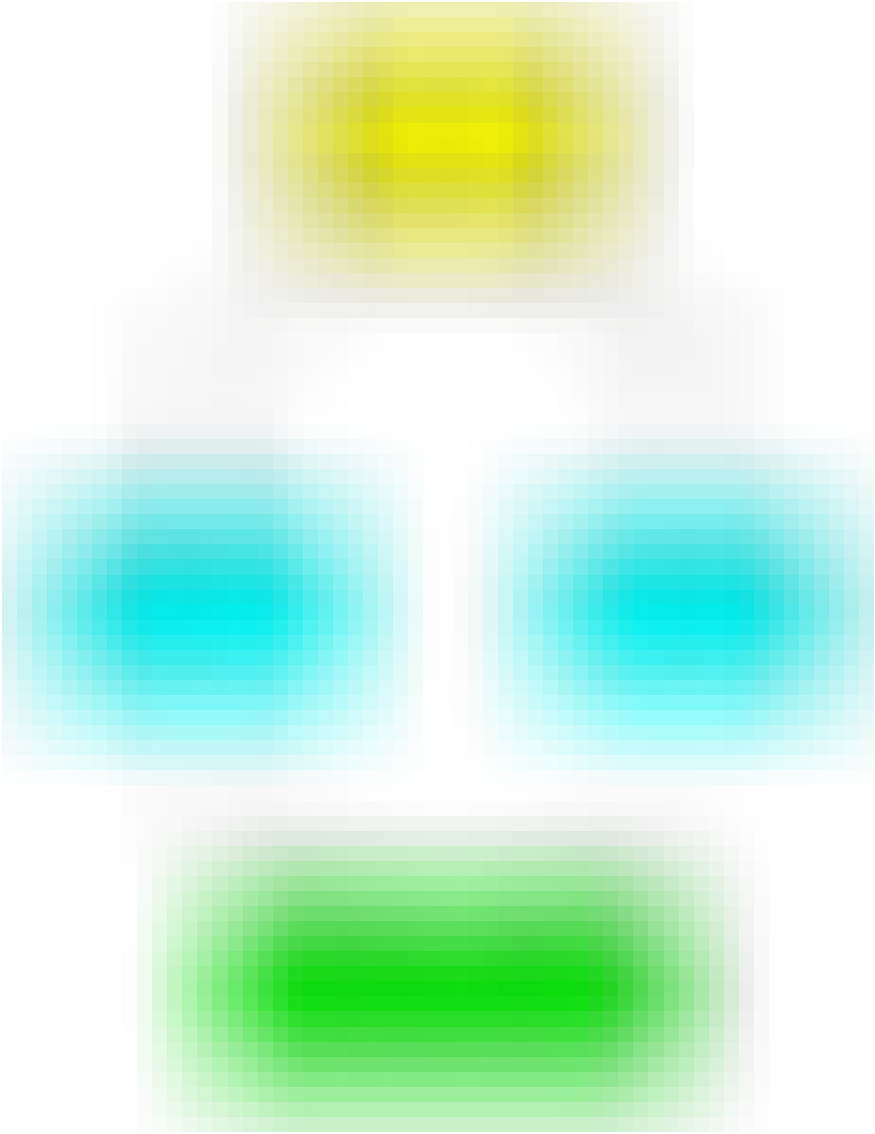
The `compose2()` function takes the `double` function as the first argument, the `inc` function as the second, and then applies the composition of those two functions to the argument `3`. Looking at the signature of `compose2()` again, `f` is `double()`, `g` is `inc()`, and `x` is `3`. The function call, `compose2(double)(inc)(3)`, is actually 3 different function invocations:

1. The first passes `double` and returns a new function.
2. The returned function takes `inc` and returns a new function.
3. The next returned function takes `3` and evaluates `f(g(x))`, which is now `double(inc(3))`.
4. `x` evaluates to `3` and gets passed into `inc()`.
5. `inc(3)` evaluates to `4`.
6. `double(4)` evaluates to `8`.
7. `8` gets returned from the function.

When software is composed, it can be represented by a graph of function compositions. Consider the following:

```
append = s1 => s2 => s1 + s2
append('Hello, ')('world!')
```

You could represent it visually:



Lambda calculus was hugely influential on software design, and prior to about 1980, many very influential icons of computer science were building software using function composition. Lisp was created in 1958, and was heavily influenced by lambda calculus. Today, Lisp is the second-oldest language that's still in popular use.

I was introduced to it through AutoLISP: the scripting language used in the most popular Computer Aided Design (CAD) software: AutoCAD. AutoCAD is so popular, virtually every other CAD application supports AutoLISP so that they can be compatible. Lisp is also a popular teaching language in computer science curriculum for three reasons:

1. Its simplicity makes it easy to learn the basic syntax and semantics of Lisp in about a day.

2. Lisp is all about function composition, and function composition is an elegant way to structure applications.

3. The best computer science text book I know of uses Lisp: Structure and Interpretation of Computer Programs.

The Fall of Composable Software

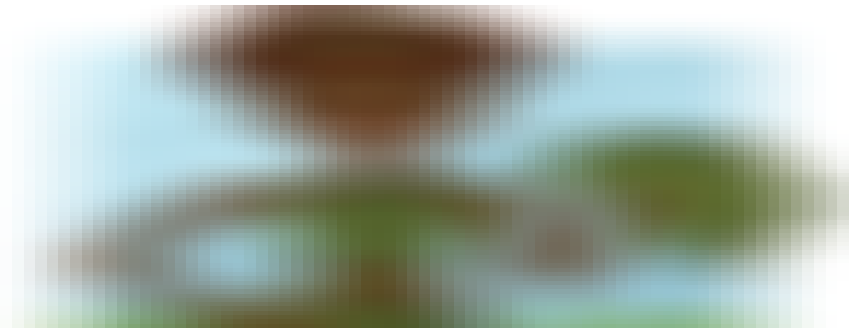
Somewhere between 1970 and 1980, the way that software was created drifted away from simple compositions, and became a list of linear instructions for the computer to follow. Then came object-oriented programming—a great idea about component encapsulation and message passing that got distorted by popular languages into a horrible idea about inheritance hierarchies and **is-a** relationships for feature reuse.

Functional programming was relegated to the sidelines and academia: The blissful obsession of the geekiest of programming geeks, professors in their ivy league towers, and some lucky students who escaped the Java force-feeding obsession of the 1990's—2010's.

For most of us, creating software was a bit of a nightmare for 30 years. Dark times.

The Rise of Composable Software

Around 2010, something great began to happen: JavaScript exploded. Before about 2006, JavaScript was widely considered a toy language used to make cute animations happen in web browsers, but it had some powerful features hidden in it. Namely, the most important features of lambda calculus. People started whispering in the shadows about this cool new thing called "functional programming".



By 2015, the idea of building software with function composition was popular again. To make it simpler, the JavaScript specification got its

first major upgrade of the decade and added arrow functions, which made it easier to create and read functions, currying, and lambda expressions.

Arrow functions were like rocket fuel for the functional programming explosion in JavaScript. Today it's rare to see a large application which doesn't use a lot of functional programming techniques.

Composition is a simple, elegant, and expressive way to clearly model the behavior of software. The process of composing small, deterministic functions to create larger software components and functionality produces software that is easier to organize, understand, debug, extend, test, and maintain.

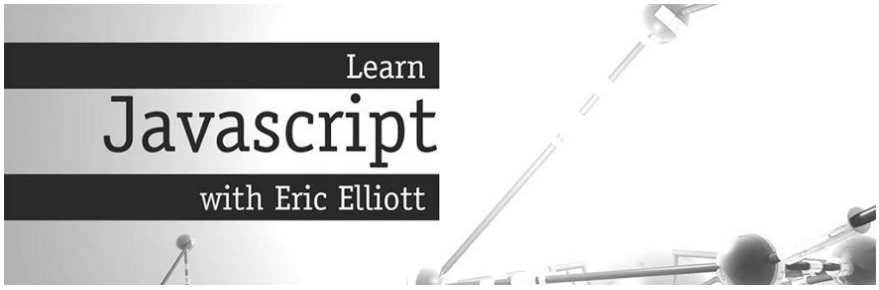
As you read the following text, please experiment with the examples. Remember what it was like as a child to pick things apart, explore, and play while you learn. Rediscover the childhood joy of discovery. Let's make some magic.

Continued in part 2: "Why Learn Functional Programming in JavaScript?"

Next Steps

Want to learn more about functional programming in JavaScript?

Learn JavaScript with Eric Elliott. If you're not a member, you're missing out!



. . .

Eric Elliott *is the author of "Programming JavaScript Applications" (O'Reilly), and "Learn JavaScript with Eric Elliott". He has contributed to software experiences for* ***Adobe Systems****,* ***Zumba Fitness****,* ***The Wall Street Journal****,* ***ESPN****,* ***BBC****, and top recording artists including* ***Usher****,* ***Frank Ocean****,* ***Metallica****, and many more.*

He spends most of his time in the San Francisco Bay Area with the most beautiful woman in the world.