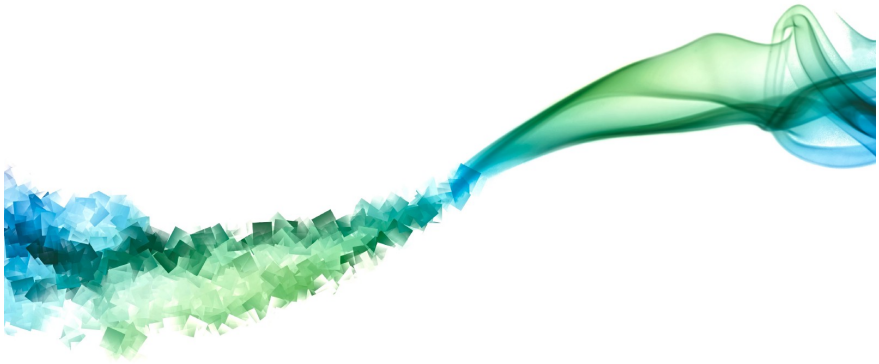
Eric Elliott Follow
Compassionate entrepreneur on a mission to end homelessness.
Mar 3, 2017 · 5 min read

Higher Order Functions (Composing Software)



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

> *Note: This is part 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!*
> *< Previous* | *<< Start over at Part 1* | *Next >*

A **higher order function** is a function that takes a function as an argument, or returns a function. Higher order function is in contrast to first order functions, which don't take a function as an argument or return a function as output.

Earlier we saw examples of `.map()` and `.filter()`. Both of them take a function as an argument. They're both higher order functions.

Let's look at an example of a first-order function which filters all the 4-letter words from a list of words:

```
const censor = words => {
  const filtered = [];
  for (let i = 0, { length } = words; i < length; i++) {
```

```
    const word = words[i];
    if (word.length !== 4) filtered.push(word);
  }
  return filtered;
};
```

```
censor(['oops', 'gasp', 'shout', 'sun']);
// [ 'shout', 'sun' ]
```

Now what if we want to select all the words that begin with ‘s’? We could create another function:

```
const startsWithS = words => {
  const filtered = [];
  for (let i = 0, { length } = words; i < length; i++) {
    const word = words[i];
    if (word.startsWith('s')) filtered.push(word);
  }
  return filtered;
};
```

```
startsWithS(['oops', 'gasp', 'shout', 'sun']);
// [ 'shout', 'sun' ]
```

You may already be recognizing a lot of repeated code. There’s a pattern forming here that could be abstracted into a more generalized solution. These two functions have a whole lot in common. They both iterate over a list and filter it on a given condition.

Both the iteration and the filtering seem like they’re begging to be abstracted so they can be shared and reused to build all sorts of similar functions. After all, selecting things from lists of things is a very common task.

Luckily for us, JavaScript has first class functions. What does that mean? Just like numbers, strings, or objects, functions can be:

- Assigned as an identifier (variable) value
- Assigned to object property values
- Passed as arguments

- Returned from functions

Basically, we can use functions just like any other bits of data in our programs, and that makes abstraction a lot easier. For instance, we can create a function that abstracts the process of iterating over a list and accumulating a return value by passing in a function that handles *the bits that are different*. We'll call that function the *reducer:*

```
const reduce = (reducer, initial, arr) => {
  // shared stuff
  let acc = initial;
  for (let i = 0, { length } = arr; i < length; i++) {

    // unique stuff in reducer() call
    acc = reducer(acc, arr[i]);

  // more shared stuff
  }
  return acc;
};

reduce((acc, curr) => acc + curr, 0, [1,2,3]); // 6
```

This `reduce()` implementation takes a reducer function, an initial value for the accumulator, and an array of data to iterate over. For each item in the array, the reducer is called, passing it the accumulator and the current array element. The return value is assigned to the accumulator. When it's finished applying the reducer to all of the values in the list, the accumulated value is returned.

In the usage example, we call reduce and pass it the function, `(acc, curr) => acc + curr`, which takes the accumulator and the current value in the list and returns a new accumulated value. Next we pass an initial value, `0`, and finally, the data to iterate over.

With the iteration and value accumulation abstracted, now we can implement a more generalized `filter()` function:

```
const filter = (
  fn, arr
) => reduce((acc, curr) => fn(curr) ?
```

```
    acc.concat([curr]) :
    acc, [], arr
  );
```

In the `filter()` function, everything is shared except the `fn()` function that gets passed in as an argument. That `fn()` argument is called a predicate. A **predicate** is a function that returns a boolean value.

We call `fn()` with the current value, and if the `fn(curr)` test returns `true`, we concat the `curr` value to the accumulator array. Otherwise, we just return the current accumulator value.

Now we can implement `censor()` with `filter()` to filter out 4-letter words:

```
const censor = words => filter(
  word => word.length !== 4,
  words
);
```

Wow! With all the common stuff abstracted out, `censor()` is a tiny function.

And so is `startsWithS()`:

```
const startsWithS = words => filter(
  word => word.startsWith('s'),
  words
);
```

If you're paying attention, you probably know that JavaScript has already done this abstraction work for us. We have the `Array.prototype` methods, `.reduce()` and `.filter()` and `.map()` and a few more for good measure.

Higher order functions are also commonly used to abstract how to operate on different data types. For instance, `.filter()` doesn't have

to operate on arrays of strings. It could just as easily filter numbers, because you can pass in a function that knows how to deal with a different data type. Remember the `highpass()` example?

```
const highpass = cutoff => n => n >= cutoff;
const gt3 = highpass(3);
[1, 2, 3, 4].filter(gt3); // [3, 4];
```

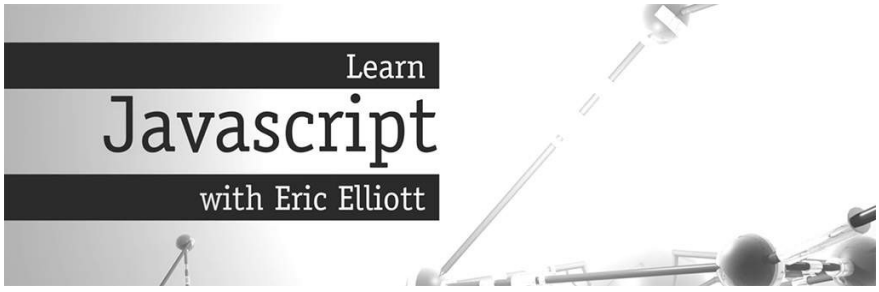
In other words, you can use higher order functions to make a function polymorphic. As you can see, higher order functions can be a whole lot more reusable and versatile than their first order cousins. Generally speaking, you'll use higher order functions in combination with very simple first order functions in your real application code.

Continue to "Reduce" >

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.