Make Your JavaScript Functionally Tasty with Currying

•••

Scott McAllister
@stmcallister

Why Functional?

"As **software** becomes more and more **complex**, it is more and more important to **structure it well**. Well-structured software is **easy to write** and to **debug**, and provides a collection of **modules** that can be **reused** to reduce future programming costs."

-- John Hughes,

Why Functional Programming Matters

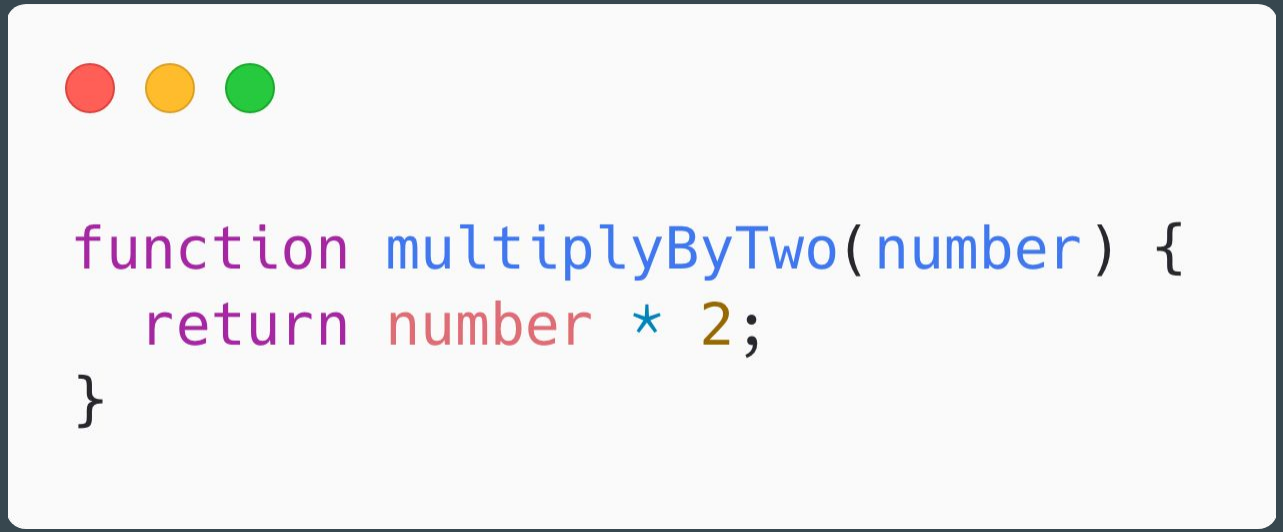
We spend more time reading code than writing it

Functional Programming Characteristics

- Pure functions
- Function composition
- Avoid shared state
- Avoid mutating state
- Avoid side effects

Pure Functions

- Given same inputs, always returns same output
- No side-effects



```
function multiplyByTwo(number) {
  return number * 2;
}
```

Function Composition

- Process of combining two or more functions to produce a new function

Shared State

- Any variable, object or memory space that exists in a shared scope

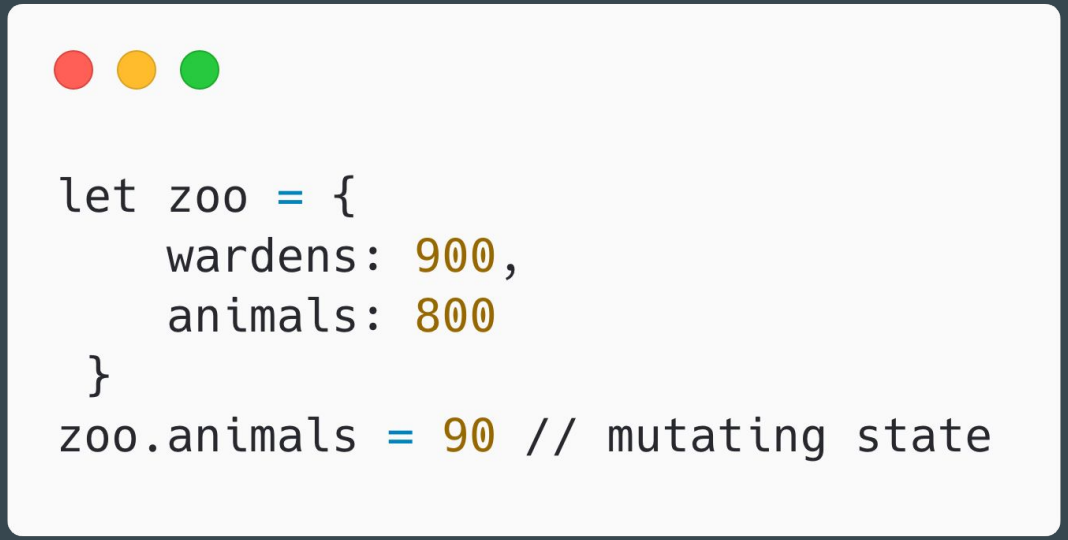
```
let x = 2;

const add = (y) => {
  x += y;
};

add(4); // x === 6 (the first time)
```

Mutating State

- When an object is changed



```
let zoo = {
    wardens: 900,
    animals: 800
 }
zoo.animals = 90 // mutating state
```

Side Effects

- Any state changes observable outside the called function other than its returned value, including:
  - Modifying external variable or object value
  - Logging/Writing to console, screen, file or network
  - Triggering any external process
  - Calling other functions with side-effects

Pure Functions

Shared State

Mutating State

Side Effects

Pure Functions

What is Currying?

@stmcallister

What is Currying?

@stmcallister

What is Currying?

@stmcallister

What is Currying?

@stmcallister

What is Currying?

@stmcallister

What is Currying?

@stmcallister

What is Currying?

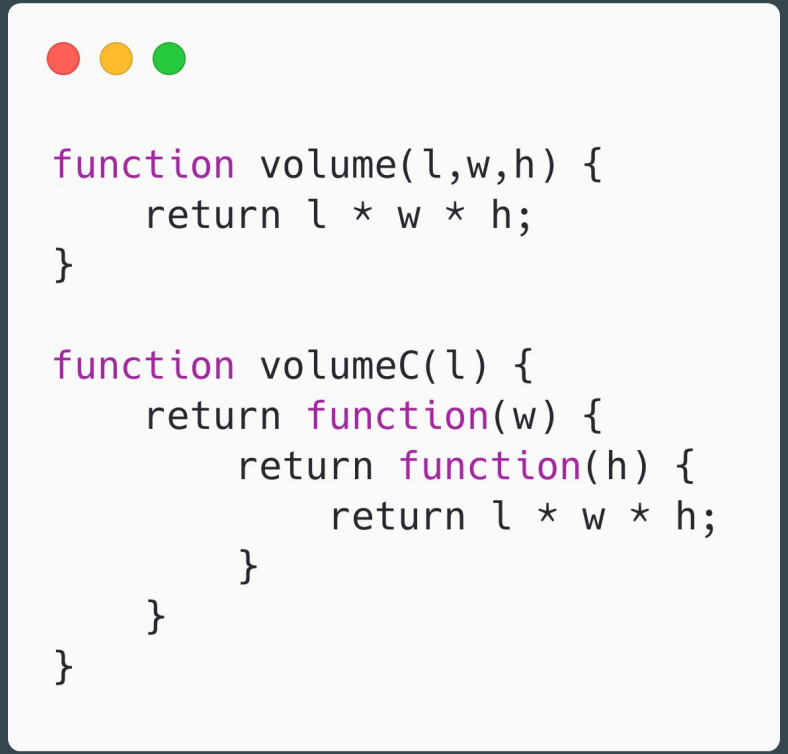
What is Currying?

Haskell Brooks Curry

Currying

Currying is a process in functional programming where **functions** with **multiple arguments** are transformed into a **series of nested single-argument** functions

Currying



```
function volume(l,w,h) {
    return l * w * h;
}

function volumeC(l) {
    return function(w) {
        return function(h) {
            return l * w * h;
        }
    }
}
```

Why Currying in JavaScript is Possible

- Closures
- Higher-order Functions

Closure

The combination of a function and its reference to its environment--which allows it to access contents of that environment

Closure

```
function init() {
  var name = 'Seattle Code Camp'; // name is a local variable created by init
  function displayName() { // displayName() is the inner function, a closure
    alert(name); // use variable declared in the parent function
  }
  displayName();
}
init();
```

@stmcallister

Closure Scopes

- Local scope
- Outer scope
- Global Scope

```
// global scope
var e = 10;
function sum(a){
  return function(b){
    return function(c){
      // outer functions scope
      return function(d){
        // local scope
        return a + b + c + d + e;
      }
    }
  }
}

console.log(sum(1)(2)(3)(4)); // log 20
```

@stmcallister

Functions are First Class Citizens in JavaScript

(Meaning, functions can be passed in as arguments to other functions

and returned as data)

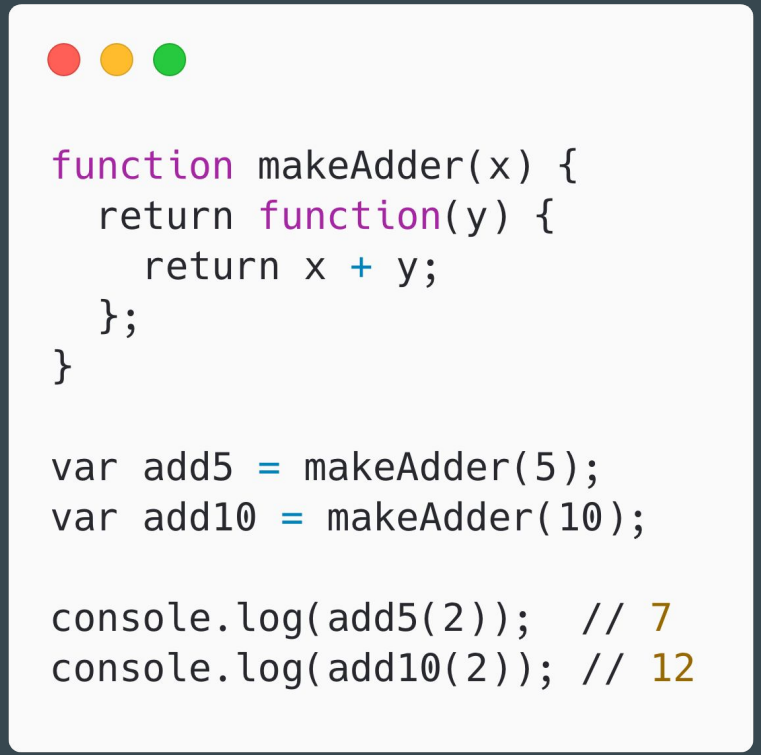
@stmcallister

Higher-order Functions

“A higher-order function is a function that takes one or more functions as arguments or a function that returns a function”

--Closurebridge

Partial Application



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

var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

Why Currying?

"A programmer's pipe-dream is to write code, and be able to use it repeatedly with little effort. It's expressive because you write in a way that expresses what is needed, and it's reuse because.. well, you're reusing. What more could you want?"

-- Hugh FD Jackson

Why Currying

The code is organized into **little specialized pieces** that can be **reused** with ease and without clutter

We spend more time reading code than writing it

Scott McAllister

stmcallister.github.io

@stmcallister

PagerDuty