Seena Rowhani

Follow 37 Followers About

🚀 Infinite Currying in Javascript

Seena Rowhani Dec 18, 2018 · 2 min read

Photo by Caroline Attwood on Unsplash

Javascript is an interesting language with a lot of quirks. Although currying isn't a concept tied to Javascript in any way, it allows for you to implement it in some clean and concise ways.

To start, I'll go over what currying is. *Currying* is essentially a technique for partial evaluations. Depending on how the function is invoked, the effect changes. Basically meaning that the context it's used in effects the result of it.

Take for example, a simple function that produces the sum of three numbers. By currying the function, we can leave it in a state of partially evaluated until all variables are provided.

```

// sum(1, 2, 3) => 6
// sum(1)(2)(3) => 6
```

basic_currying.js hosted with ❤ by GitHub view raw

As seen above `curriedSum` returns a function keeping the previously passed in parameters in its context. So you'd be able to partially evaluate this function, and pass the function around to be used in other places. Cool right!

Now I'll introduce the concept of infinite currying. In Javascript, every object can implement this method `valueOf` that when the object is evaluated, the result would be the result of the method call.

Also in Javascript, everything is an object (besides primitives) — including functions!

```
function sum (...args) {
  return Object.assign(
    sum.bind(null, ...args),
    { valueOf: () => args.reduce((a, c) => a + c, 0) }
  )
}

// console.log(+add(1)(2)(3)(1, 2, 3)) = 12
```

curry.js hosted with ❤ by GitHub view raw

So here we see a rewrite of the above function, but allows for infinite currying. Let's break it down. Since a function is also an object, we can assign new properties onto it!

The usage of Object.assign here allows us to return a new function instance with bound arguments. When you bind arguments to a function, they are embedded into the function call essentially.

```
const sum = (a, b, c) => a + b + c
const boundSum = sum.bind(null, 1, 2)

// console.log(boundSum(3)) => 6!
```

binding_example.js hosted with ❤ by GitHub view raw

The first argument of bind represents the context of the function call (i.e what `this` will represent inside of the function. Since we have no use for `this`, we can just pass in null.

To get back to the original example, we were performing:

```
  { valueOf: () => args.reduce((a, c) => a + c, 0) }
)
```

So we know that calling bind on sum will return a new sum function with the old arguments embedded into it. But we haven't said much about the next bit.

Object.assign here will take all the properties of the second object, and place them into the first object. In this case, our first object being our function. So the valueOf we've defined will return the sum of all the available arguments given at when called.

```
const partiallyEvaluated = sum(1, 2, 3)(4)(5, 6)
console.log(+partiallyEvaluated) // => 21
console.log(Number(partiallyEvaluated)) // => 21
```

And that's it! Try it for yourself 🎉.

JavaScript Curying