Javascript- Currying VS Partial Application

Deepak Gupta
Jul 22, 2018 · 5 min read ★

A lot of people get confused in between currying and partial application and many of us does not know what, where and when we should use them. So this post will cover the practical usage and difference in them.

So lets's start with definitions

· · ·

Currying

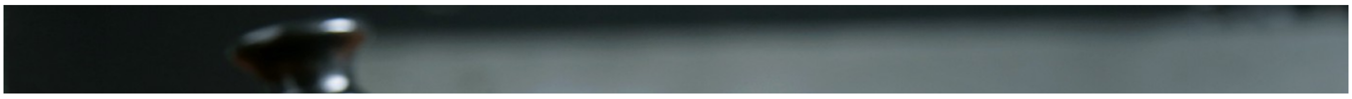


Photo by Caroline Attwood on Unsplash

Is a technique for converting function calls with N arguments into chains of N function calls with a single argument for each function call.

Currying always returns another function with only one argument until all of the arguments have been applied. So, we just keep calling the returned function until we've exhausted all the arguments and the final value gets returned.

```
// Normal function
function addition(x, y) {
   return x + y;
}

// Curried function
function addition(x) {
   return function(y) {
     return x + y;
   }
}
```

> *Note: Curry takes a binary function and returns a unary function that returns a unary function. Its JavaScript code is*

```
function curry(f) {
  return function(x) {
    return function(y) {
       return f(x, y);
```

```
      }
    }
  }
```

> *Note: A curried functions have a built-in iterator behaviour. One argument is applied at once which is then returned to the calling function to be used for next step. Read here about iterators.*

Usages

1. The common use-case for curried function are **function composition**.e.g., `p(x) = q(r(x))` . i.e building new function from old function by passing arguments. Function `q` takes the return value as an argument from function `r` . Since a function can only return one value, the function being applied to the return value must be unary.

2. Curried function can also be used while **infrastructure setup** of a project where there is a lot of possibility to create generic functions thereby little pieces can be configured and reused with ease, without clutter.

3. **Ramda.js** lib. functions are automatically curried and lodash has a function called curry which can be used to form curry function.

4. **Memoization** is another good use case for curry function.

5. **Handling error** throwing functions and exiting immediately after an error.

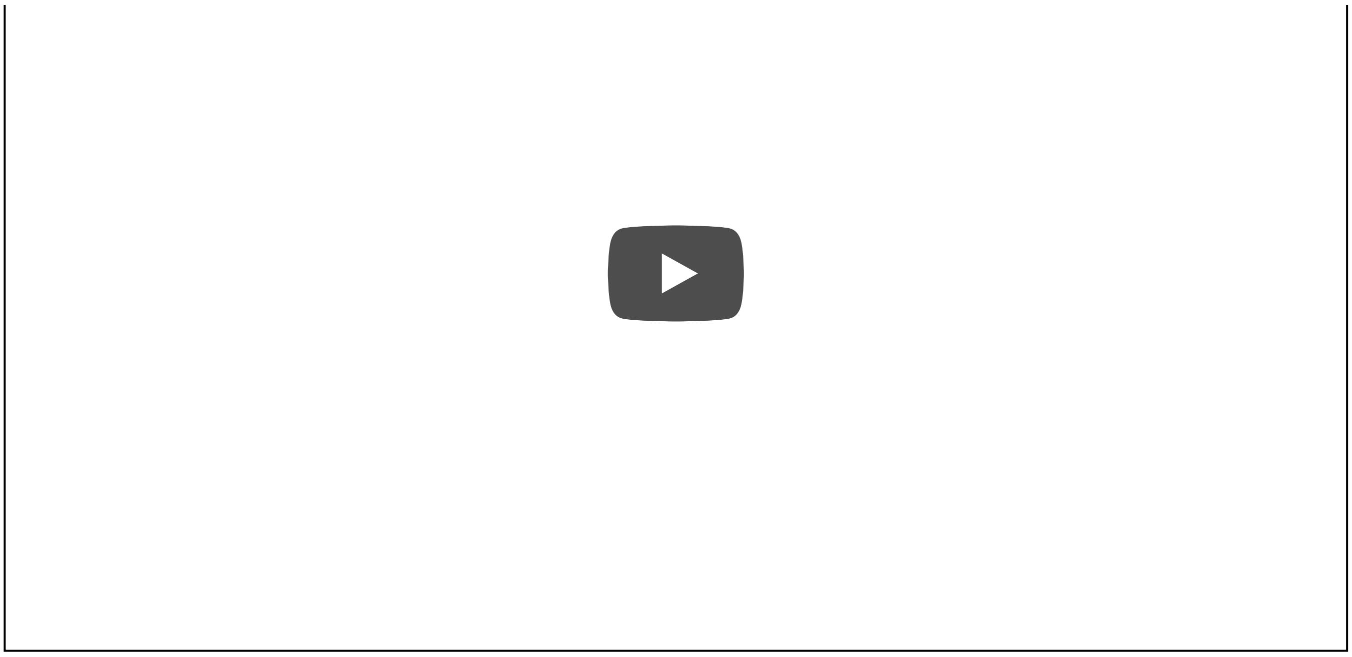
6. **Catching multiple error and use it as a validator** on API's and client side code.

7. Can create **First class functions** which means that we can use functions as arguments and return values. Eg:

```
const func1 = () => console.log ('Hey Medium.');
const firstClassfunc1 = argsFunc => argsFunc();
const firstClassfunc2 = () => func1;
firstClassfunc1 (firstClassfunc2()); // Hey Medium.
```

> *Note: Do go through this video and you know more about why, when and usage.*

Hey Underscore, You're Doing It Wrong!



Curry Factory Method(ES6)

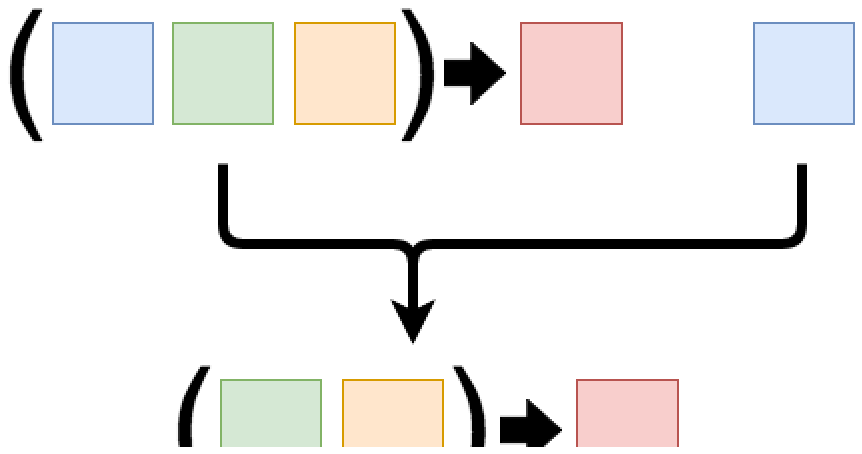
```
const compose = (...fns) =>
  fns.reduce((f, g) => (...args) => f(g(...args)));
```

Here is a good article on how to make our own curry function factory method using ES5.

> *Note: Currying in Javascript if not done properly can leads to more complicated stack traces, which is unfortunate for debugging.*

. . .

Partial Application



Is a technique of fixing a number of arguments to a function, producing another function of smaller arguments i.e binding values to one or more of those arguments as the chain of function progressed.

```
function add1(x) {
  return 1 + x;
}
```

JavaScript has the built-in method `.bind` that works on functions with any number of arguments and can bind an arbitrary amount of parameters. Its invocation has the following syntax.

```
function.bind(thisValue, [arg1], [arg2], ...)
```

It turns *function* into a new function whose implicit this parameter is this value and whose initial arguments are always as given.

```
function addition(x, y) {
   return x + y;
}

const plus5 = addition.bind(null, 5)
plus5(10) // output -> 15
```

> *Note:* `this` *value does not matter for the (non-method) function addition which is why it is null above.*

When using underscore or lodash you can use the *partial* function which is much nicer than the raw *bind* method.

Here is a detailed post on partial application and left, right partial application function implementation.

. . .

Difference

- Currying *always* produces nested unary (1-ary) functions. The transformed function is still largely the same as the original.
- Partial application produces functions of arbitrary number of arguments. The transformed function is different from the original — it needs less arguments.
- Currying is not partial application. It can be implemented using partial application. You can't curry a function that takes any number of arguments, (unless you fix the number of arguments).

Advantages

Easier reuse of more abstract functions which leads to clean code and less complexity which improves code expressive power and maintainability.

. . .

Please consider **entering your email here** if you'd like to be added to my email list and **follow me on medium to read more article on javascript and on github to see my crazy code**. If anything is not clear or you want to point out something, please comment down below.

Tips Are Appreciated! 💰 😉
My Bitcoin address: 132Ndcy1ZHs6DU4pV3q2X1GzSCdBEXX6pz

My Ethereum address: 0xc46204dfc8449Ffb0f02a9e1aD81F30D3f027010

. . .

You may also like my other articles

1. Nodejs app structure
2. Javascript — Generator-Yield/Next & Async-Await
3. Javascript data structure with map, reduce, filter

4. Javascript ES6 — Iterables and Iterators

5. Javascript performance test — for vs for each vs (map, reduce, filter, find).

6. Javascript — Proxy

. . .

If you liked the article, please clap your heart out. Tip — You can clap 50 times! Also, recommend and share to help others find it!

THANK YOU!

codeburst.io

JavaScript Technology Web Development Functional Programming Code