Currying

Técnica muito importante em Programação Funcional. Sua nomenclatura advém do matemático *Haskell Brooks Curry*, criador da linguagem de programação funcional Haskell, inclusive!

"Currificar" uma função significa reorganizar sua escrita para que a passagem de parâmetros seja definida gradualmente.

```
1  // Versão padrão
2  function op(a,b,c) {
3      return a * (b - c)
4  }
5  console.log(op(4,3,1))
6
7  // Versão "curried"
8  function subcurried(a) {
9      return function (b) {
10         return function (c) {
11             return a * (b-c)
12         }
13     }
14 }
15 console.log(subcurried(4)(3)(1))
```

Uma das enormes vantagens de se utilizar versões *curried* de funções é fomentar o **REUSO** através da viabilização da **APLICAÇÃO PARCIAL**. Já vimos muitos exemplos disso anteriormente no curso.

```
// Versão "curried" com notação arrow.
const subcurried = (a) => (b) => (c) => a * (b - c)
console.log(subcurried(4)(3)(1))

// Reuso através da aplicação parcial da função.

//Ex. função subtrair dois números
const subtrair = subcurried(1)
console.log(subtrair(5)(8))

//Ex. função dobro de um número
const dobro = subcurried(-2)(0)
console.log(dobro(10))

//Ex. função negativo de um número
const negativo = subcurried(1)(0)
console.log(negativo(11))
```