Higher Order Functions

What makes a function higher order?

Well, just that it takes another function as an argument. You see these functions think that they are higher because they use other functions to do their dirty work. Really, these functions are just bossy.

Let's have a look at some examples:

```
higher :: (a -> b) -> [a] -> [b]
notHigher :: a -> b
definitelyNotHigher :: iAmJustAHumbleValue
theHighest :: (b -> c) -> (a1 -> a2 -> b) -> a1 -> a2 -> c
palpatine :: (order -> chaos) -> (jedi -> death) -> (younglings -> egregiousDeath) -> thePowerOfT
```

First Class

In Haskell, function are first class.

I know what you are thinking: "Well of course Haskell functions are first class! Haskell functions are the bomb. Couldn't have described them better myself :-D", and I 100% agree. However, when I say that functions in Haskell are first class I mean this as a technical term. Not that they are simply the best (the fact that they are is obvious enough). Not even that they have bought a fancy plane ticket. In programming, something being first class means that it can be considered as a value. So in Haskell, functions being first class means that they are all values, and being values is why they can be passed to functions. This is why higher order functions are a thing.

TLDR; Programming terminology says: First Class = something can be considered and used as a value.

What is the big deal?

First class functions, and in turn higher order functions is one of the things that sets Haskell and the whole functional programming paradigm apart. They are an extremely useful tool of abstraction. They make it super easy to to the same thing to every element in a list (map), in fact they make it easy edit all elements within some arbitrary structure (fmap). With them we can also generally crush structures into a single representative value (folds), or morph one structure into another.

Higher order functions are basically the master of the black box. Black boxes are fab because you don't need to know how they work, you just need to know what they do, then you can build upon them, without having to worry about their details, to create something even cooler! This is abstraction. Higher order functions are a key example of abstraction in Haskell, and abstraction is what functional programming is all about: forgetting the annoying details and focusing on the bigger picture. You are not some C engineer having to have your fingers in every pie to make everything run smoothly; you are an orchestra conductor. You may not be able to play every instrument, but you understand its value and how it contributes to the piece as a whole. The land of haskell is abundant with cool functions that you can discover on Hoogle. They are all at your disposal if you learn the ways of composition and using higher order functions. Be the conductor!

Higher order functions is just one of the many functional programming ideas that other languages have copied. Java 8 brought in its own way of using higher order functions because it saw the light and how useful it was. Sadly, while the functionality is there, their way of using higher order functions is clunky and ew. Like any remake, it is just not as good as the original. Don't even get me started on how C allows higher order functions using POINTERS. Higher order functions are native to functional programming and so that is where they are best presented.