Fmap and map functions in haskell

Introduction

Haskell is an advanced, statically typed, lazy, concurrent, purely functional programming language. Among other features, high order functions are provided by Haskell by default.

Two of those high order functions can cause beginners confusion and even give experts discussion about merging them and using one instead of the other: map && fmap.

Map

The map function returns a list created by applying a function in all elements of a second list passed as parameter, the syntax is:

map :: (a -> b) -> [a] -> [b]

That means we can create a function that creates a b given an a and using map apply that function to all a list of a’s generating a new list of b’s.

Examples:

Prelude> map (+3) [1,2,3]

[4,5,6]

Prelude> map reverse [“Lazy”, “Programming”, “Language”]

[“yzaL”, “gnimmargorP”, “egaugnaL”]

Fmap

The fmap function is a generalized version of map that works for lists as well but can also be used for any functor

fmap :: (a -> b) -> f a -> f b

Functor

According to Haskell documentation Functor is a ‘class used for types that can be mapped over’

When we think about mapping over first thing that comes to mind are lists, but with functors anything can be mapped over. Functors can help in all cases where you have a container for some data and want to be able to map predicates to your value.

The simplest example is Maybe:

**instance** Functor Maybe **where**

fmap f (Just x) = Just (f x)

fmap f Nothing = Nothing

This way the use of a Maybe can be abstract if you don’t want to care about dealing with the Nothing. Fmap will only do anything if we have a just value. This makes sense because what would be the behaviour of trying to make operations in nothing, what would be the outcome?

Examples:

Prelude> fmap (+3) [1,2,3]

[4,5,6]

Prelude> fmap reverse [“Lazy”, “Programming”, “Language”]

[“yzaL”, “gnimmargorP”, “egaugnaL”]

Prelude> fmap (+3) Nothing

Nothing

Prelude> fmap (+3) (Just 3)

Just 6

But this is just the tip of the iceberg, functors are present in function compositions, can be used over the IO, trees, Either constructors and can arbitrary be used for mapping anything over any desired thing.

And what this mean for us, haskellers?

Maps are a just fmaps for lists, lists have their own functor implementation. When dealing with lists we can use map or fmap, the output will be the same.

It might be convenient to use map over lists because the errors are going to be narrow and easier to follow

Fmaps are generic and can do a lot more than what I expected before really start understanding and writing about them.

<https://www.quora.com/Is-it-better-to-use-fmap-instead-of-map-in-Haskell>

<https://stackoverflow.com/questions/6824255/whats-the-point-of-map-in-haskell-when-there-is-fmap>

<https://bugfactory.io/2014/09/04/map-and-fmap-in-haskell/>

<https://wiki.haskell.org/Functor>

<http://learnyouahaskell.com/functors-applicative-functors-and-monoids>

<http://learnyouahaskell.com/making-our-own-types-and-typeclasses#the-functor-typeclass>