Build-your-own-[grep, Redis] in Haskell

Students

Olivier Lischer

Benjamin Plattner

Introduction: Often, the best ways to learn something new is to use it. Once you get past a minimum understanding of how 'this new thing' works, applying it in a real project is the key to master it.

As a software engineer you use many tools without ever really looking at the inner workings. When you embark on the adventure of learning a new programming language, this poses the ideal opportunity to apply the fresh knowledge to something you are very familiar with as a user.

This is the idea behind the build-your-own-x concept, where x stands for an application of your liking.

Objective: Many tutorials exist online which you can follow to implement your chosen tool in that new programming language you are just getting to know better. However, few of them are written for Haskell, a functional programming language known for its static and strong type system.

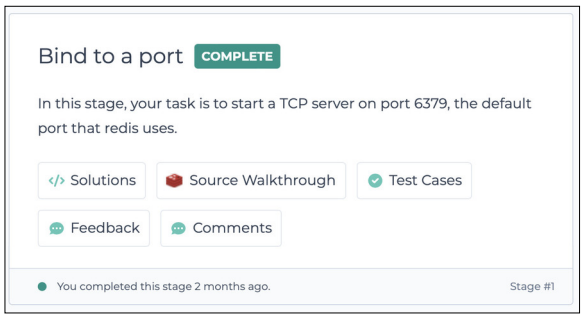
The purpose of this work is to provide detailed Haskell tutorials for two Build-Your-Own-Xs - grep and Redis.

Result: Both tutorials are fully reviewed and publicly available as open-source projects.

This open-source repository is maintained by our project partner CodeCrafters, a start-up that specializes in programming courses for experienced software engineers.

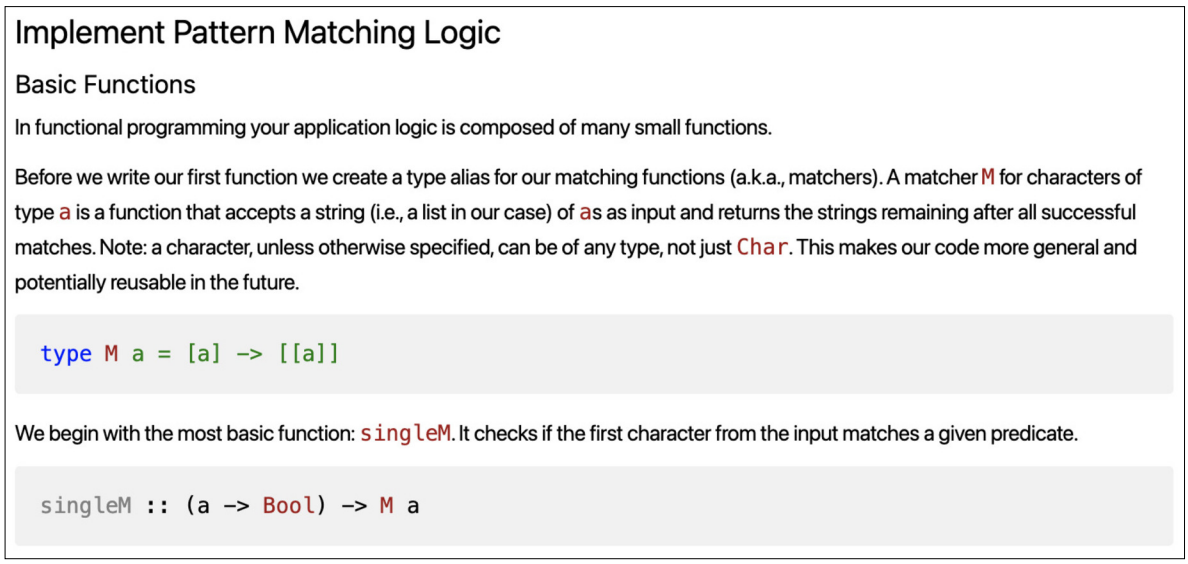
The material is also planned to be used as part of the functional programming course at the OST.

One of the stages for a user to complete at CodeCrafters - our tutorials are included in their Solutions part.
Own presentment



Submitting a solution to CodeCrafters via Command Line Interface to receive feedback on its correctness.
Own presentment

An example of the tutorial for grep: it includes detailed explanations and code references.
Own presentment



Implement Pattern Matching Logic

Basic Functions

In functional programming your application logic is composed of many small functions.

Before we write our first function we create a type alias for our matching functions (a.k.a., matchers). A matcher `M` for characters of type `a` is a function that accepts a string (i.e., a list in our case) of `a`s as input and returns the strings remaining after all successful matches. Note: a character, unless otherwise specified, can be of any type, not just `Char`. This makes our code more general and potentially reusable in the future.

```
type M a = [a] -> [[a]]
```

We begin with the most basic function: `singleM`. It checks if the first character from the input matches a given predicate.

```
singleM :: (a -> Bool) -> M a
```

Advisor
Prof. Dr. Farhad D. Mehta

Subject Area
Application Design, Software, Software Engineering - Core Systems, System Software

Project Partner
CodeCrafters.io, Sarup Banskota, 2041 East St PMB 45, Concord CA 94520, California, USA