

From setting up stack to an API client library in one evening

A Haskell Workshop

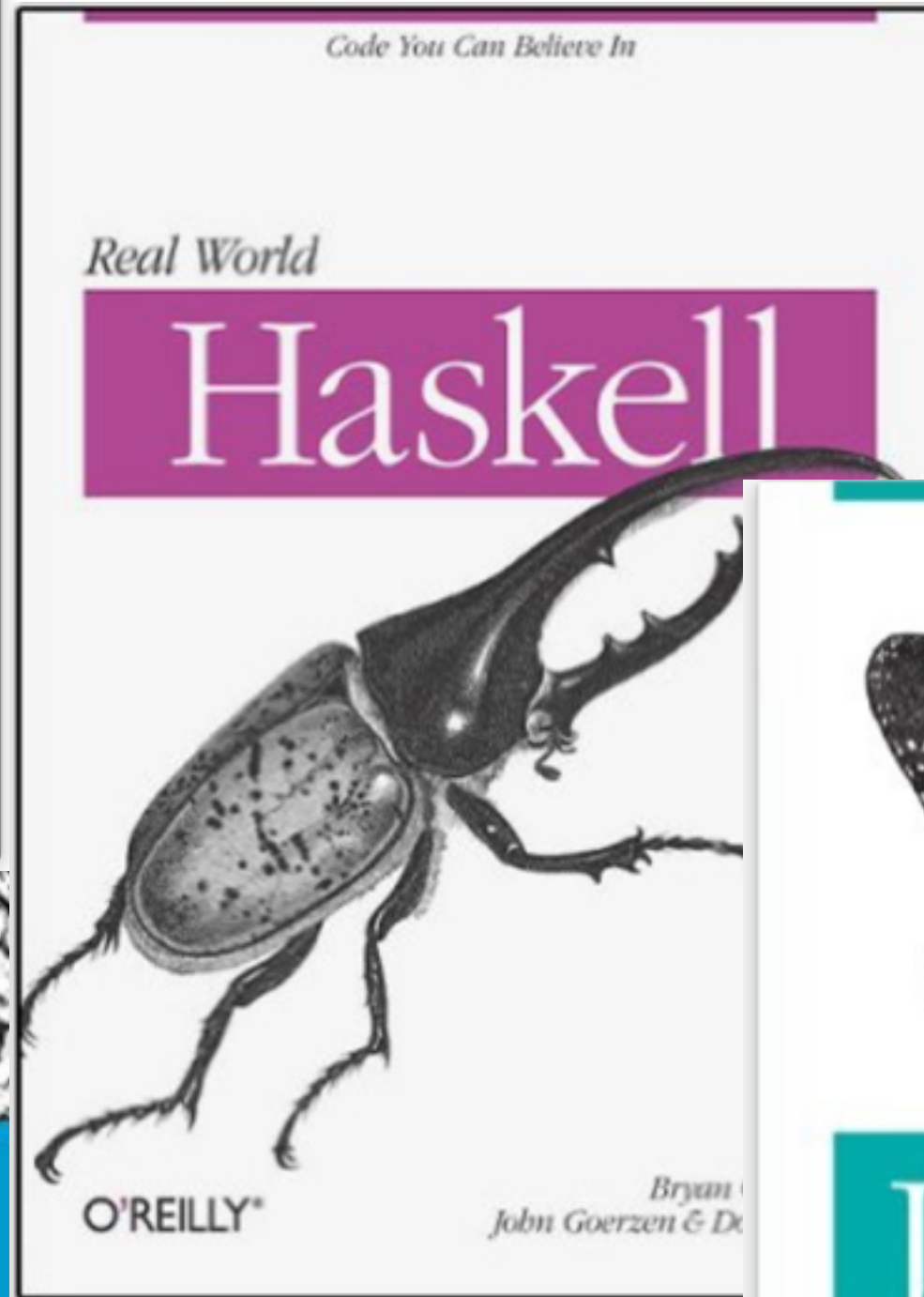
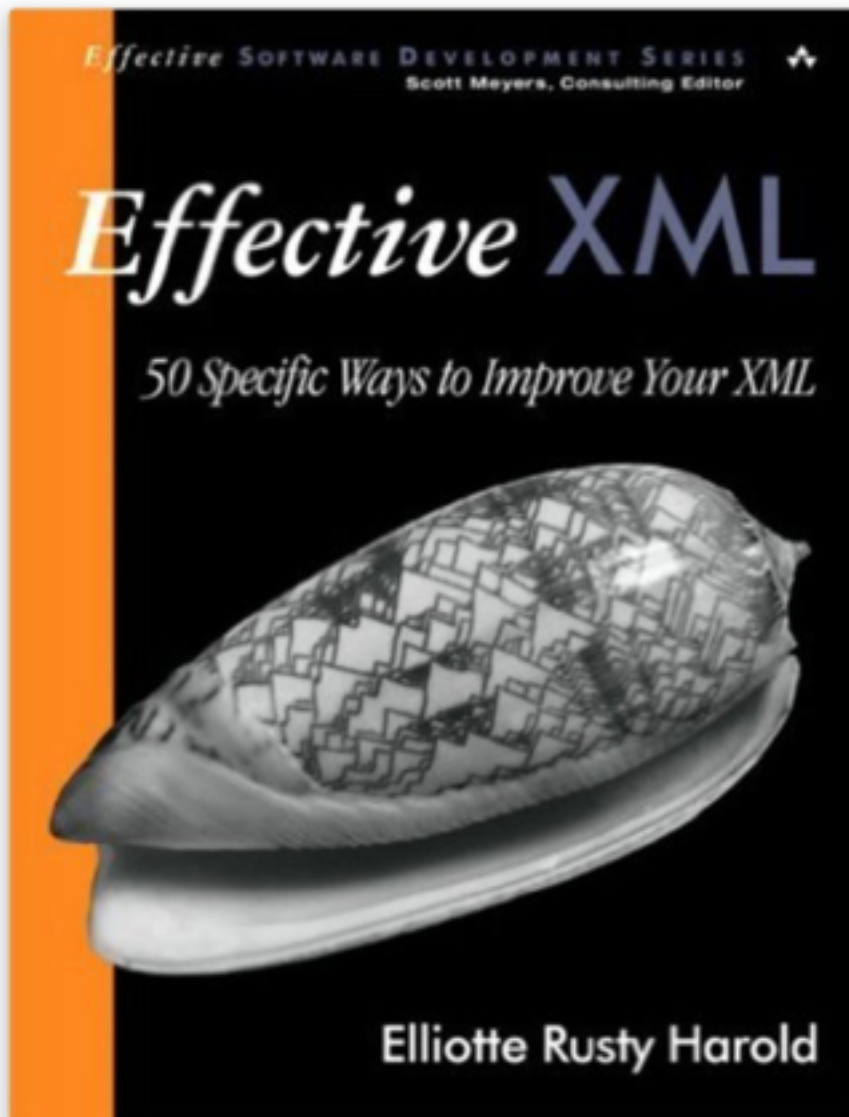
Motivation

- Haskell is too elegant to be confined to a small niche, let's not take the “Avoid success at all costs” to an extreme.
- For simple projects the initial boiler-plate can be more daunting than the code itself.
- API wrappers are easy to create and quite useful.
- As an added bonus we get to learn a bit about lenses and a tool for more general HTTP interaction.

Motivation

- So I was browsing the fiction shelves in a local bookstore and saw this...

Motivation



Stack

- Created to solve some shortcomings of the cabal toolchain, specially the “reproducible build” problem.
- It installs ghc, manages and shares dependencies, and keeps track of each project’s environment.
- LTS and Nightly repositories

Stack

```
# First let's install and take a look at the options  
  
$ brew install haskell-stack  
  
# On linux take a look at http://docs.haskellstack.org/en/stable/README.html  
  
$ stack  
  
# Now let's see what are the available templates  
  
$ stack templates  
  
# Finally we can create our project  
  
$ stack new api chrisdone
```

Stack

```
# Now we install our compiler and dependencies
```

```
$ cd api
```

```
$ stack setup
```

```
$ stack test
```

Wreq

- Official website: <http://www.serpentine.com/wreq/>
- Built on top of **http-client** and **lens**
- Session handling: keep-alive, pooling, and cookies
- Powerful multipart form and file upload handling
- **Basic** and **OAuth2** bearer authentication

Wreq Setup

- Let's give it a try.
- Edit `api.cabal`

Wreq Setup

...

```
build-depends:      base >= 4.7 && < 5
                   , wreq
                   , lens
                   , aeson
                   , lens-aeson
default-language:   Haskell2010
default-extensions: OverloadedStrings
```

...

Stack and Dependencies

- We are using packages without version constraints just for easy discovery of current stockage versions
- DO NOT publish packages like this
- After install we can edit the cabal file and inform the proper versions constraints for each package.

Wreq Quick Tour

...

```
build-depends:      base >= 4.7 && < 5
                   , wreq
                   , lens
                   , aeson
default-language:   Haskell2010
default-extensions: OverloadedStrings
```

...

Wreq Quick Tour

```
$ stack ghci
*Main Lib> import Network.Wreq
*Main Lib Network.Wreq> import Control.Lens
*Main ...> import Data.Aeon.Lens

*Main ...> let opts = defaults & param "q" .~ ["tetris"] & param
"language" .~ ["haskell"]
*Main ...> r <- getWith opts "https://api.github.com/search/repositories"
*Main ...> r ^.. responseBody . key "items" . values . key "owner" . key
"login" . _String
```

Lenses

- Official website: <http://lens.github.io>
- Here is a good intro: <http://begriffs.com/posts/2016-01-07-clear-intro-to-lenses.html>
- Intimidating at first, they completely change the code style.
- Take a look at [functional references](#)

Lenses Cheat-Sheet

Operation	Operator	Function
-----------	----------	----------

Get

$\wedge.$

view

Set

$.\sim$

set

Aeson

- JSON encoder/decoder
- Uses Type Classes FromJSON and ToJSON
- Can be used with Generics for fun and profit!

Aeson + Wreq

...

```
default-extensions: OverloadedStrings, DeriveGeneric
```

...

Aeson + Wreq

```
$ stack ghci
*Main Lib> import Network.Wreq
*Main Lib Network.Wreq> import GHC.Generics
*Main ...> import Data.Aeson

*Main ...> data Addr = Addr Int String deriving (Generic)
*Main ...> instance ToJSON Addr
*Main ...> let addr = Addr 1600 "Pennsylvania"
*Main ...> post "http://httpbin.org/post" (toJSON addr)
```

What are we coding today?

- Very simple client for Pivotal Tracker API.
- It's a project management tool based on sprints and user stories.
- API docs: <https://www.pivotaltracker.com/help/api>
- We just want to show data of a specific story.

Pivotal Tracker API

- Returns JSON documents using a simple authentication token.
- Story resource: https://www.pivotaltracker.com/help/api/rest/v5#story_resource
- Story endpoint: <https://www.pivotaltracker.com/help/api/rest/v5#Story>

You should be able to run this code after implementing your library

```
module Main where

import Network.Wreq (defaults)
import Control.Lens ((^.))
import Lib ( withToken
              , getStory
              , name
            )

main :: IO ()
main = do
    story <- getStory authentication projectId storyId
    print $ story ^. name
    where
        authentication = withToken "7f3f76bc6ae8c48e7b528369c999c8d8"
defaults
    projectId = 1440520
    storyId = 104591424
```

To run your app

```
$ stack exec api-exe
```

Too easy?

- If you finished try implementing some tests.
- You can run them using **stack test**.
- There is already a template for you in the test dir.