

A Beginner's Guide to Haskell and its Ecosystem

Alejandro Serrano @ Haskell eXchange 2021

 @trupill -  47 Degrees (Academy)

Goal

Learning a new programming environment involves many things:

- The language itself
- The build tools
- Where to find dependencies
- Good idioms and practices
- ...

Goal

Learning a new programming environment involves many things:

- ~~The language itself~~ too much focus on this
- The build tools
- Where to find dependencies
- Good idioms and practices
- ...

A bird's eye view of design and ecosystem


Contents

 What makes Haskell ~~hard~~ special

 Getting started

 Projects in Haskell

 Extensions and the Type Level

 Community



What makes Haskell
~~hard~~ special



A different beast

Haskell (and FP) have brought many ideas to the table: (list not complete)

- Higher-order functions
 - Higher-rank and impredicativity
- Algebraic data types and pattern matching
- Purity and laziness
- Type-directed contexts with type classes
 - Functors and monads everywhere
- Working with and manipulating types

Haskell's Manifesto (about design)

Define **data** in a simple way

- Enforce *invariants* at compile-time
- Powerful *pattern matching*

Use types to **guide** your program

- Be explicit about your constraints
- Compiler-based DI

Think about **side effects**

- No tag = no side effects (pure)

Haskell data is mainstream

Some ideas are nowadays in
Scala / Kotlin / Java / C# / Swift:

- Higher-order functions
- Algebraic data types
 - Records / data classes
 - Sealed hierarchies

Learn concepts in a familiar setting

Purity

Anything outside computation (**side effects**) is marked in the type:

```
(++) :: String -> String -> String  
readContents :: String -> IO String
```

This **restricts** composition:

- Forces a *pure core / effectful edges* design
- Drawing the line is difficult at first

Laziness

This function is short-circuiting:

```
and :: Bool -> Bool -> Bool
and False _ = False
and _      x = x
```

In Haskell computations are executed:

- Only when needed
- As much as they are needed

Laziness

In Haskell computations are executed:

- Only when needed
- As much as they are needed

This is hard!

- Debugging follows weird paths
- It works in the small, it leaks in the large


Getting started

Our tool manager: ghcup

Easiest way to get a working environment

Similar to SDKMAN, Coursier, `rustup`...

- > `ghcup install ghc recommended`
- > `ghcup install cabal recommended`
- > `ghcup install stack recommended`
- > `ghcup install hls recommended`

 Don't worry about versions for now

Our editor integration: HLS

Based on Language Server Protocol

- Support from VS Code to Emacs
- In VS Code, just install the *Haskell* plug-in

⚠ Sometimes HLS support lags behind

- You can check support with `ghcup list`

x	ghc	8.10.6	base-4.14.3.0	hls-powered
✓✓	ghc	8.10.7	recommended,base-4.14.3.0	hls-powered
✓	ghc	9.0.1	base-4.15.0.0	hls-powered
x	ghc	9.2.1	latest,base-4.16.0.0	

Start a new project


Not one, but two choices: (why, in just a second)

```
> cd cool_project && cabal new  
> stack new cool_project
```

Start a new project

Not one, but two choices: (why, in just a second)

```
> cd cool_project && cabal new  
> stack new cool_project
```

Another good tool is  Summoner

- Good set of default warnings
- Additons like GH Actions integration

Projects in Haskell

Cabal vs. cabal vs. .cabal

Haskell's build tool story is messy

- **Packages** are the unit of distribution
 - Package = set of modules
 - Module = source file = thing you import
- Each package comes with a **build file**
 - Dependencies, exposed modules, flags
 - In the `<package-name>.cabal` file
 - Often called *the Cabal file*

Simple Cabal file

Not JSON, not YAML, something in between

Each **stanza** defines libraries or executables

```
name:      haskell-exchange
```

```
version: 0.1.0.0
```

```
author:   Alejandro Serrano
```

```
executable talk
```

```
  main-is:      Main.hs
```

```
  build-depends: base ^>= 4.12, aeson ^>= 2
```

```
  hs-source-dirs: app
```

```
test-suite talk-tests
```

```
  ...
```

Build tools: Cabal and Stack

You can **build** such package in two ways:

1. Cabal: `cabal build`
2. Stack: `stack build`
 - May require `stack init`

Build tools: Cabal and Stack

You can **build** such package in two ways:

1. Cabal: `cabal build`
2. Stack: `stack build`

The choice is nuanced, but in general terms:

- Stack focuses on reproducibility by default
- Cabal supports reproducible tools via `freeze`
- Stack tracks your toolchain (installs GHC)
- Cabal doesn't, but we have `ghcup` now

Dependencies: Stackage and Hackage

Stacks requires an additional `stack.yaml` file

```
resolver: lts-18.14 # package set
packages:
- .                # your project
```

A **resolver** defines a set of fixed packages and their versions known to build together

- Haskell packages are built from source
- Because of strong typing, this is a **huge deal**

LTS targets the "recommended" GHC version

- new minor version of LTS only updates minor version of the packages

Nightly targets one version more

- more recent version of packages
- at some point, they "graduate" to LTS

What about the **most** recent version?

Hackage - `hackage.haskell.org`

The repository for Haskell packages

All version of packages + their documentation

Hackage - hackage.haskell.org

The repository for Haskell packages

All version of packages + their documentation

You can add Hackage packages to Stack

```
resolver: lts-18.14
packages:
- .
extra-deps:
- nice-dependency-0.24.5
```



Summary

- Packages are defined in `.cabal` files
- Cabal and Stack are used to build them
- We have two sources for dependencies:
 - Stackage for curated sets
 - Hackage for everything
- Stack leans towards Stackage



Extensions and the Type Level

Haskell-the-language

The language itself vs. the compilers

There is more than one Haskell compiler:

- GHC, from Glasgow
- Helium and UHC, from Utrecht
- Mu, used internally
- LHC, JHC, ...

Haskell-the-language

The language itself vs. the compilers

There is more than one Haskell compiler:

- **GHC** is *de facto* the standard
- *Helium*, in research, and UHC
- *Mu*, used internally
- LHC, JHC, ..., not maintained

Haskell-the-language

The language itself vs. the compilers

Report = Haskell "standard"

- Current: Haskell 2010
- Previously: Haskell 98

Haskell-the-language

The language itself vs. the compilers

Report = Haskell "standard"

- Current: Haskell 2010
- Previously: Haskell 98

*You sometimes hear people saying they write
"Haskell 2010" (or 98) code to mean code
without GHC extensions*

Extensions

Anything outside the Report

- Syntactical goodies
- Type classes with multiple parameters
- Type families
- More ways to use `deriving`
- And other 20+ things

 Extension `/=` unsupported or bad style

Extensions

Anything outside the Report

To enable one, you write **first thing** in the file

```
{-# language MultiParamTypeClasses #-}  
-- this also works, but no need to shout 💡  
{-# LANGUAGE DeriveFunctor #-}
```

Usual joke: a Haskell file is 20 lines extensions,
30 lines type definitions, and 3 lines of code

Type level

Working with types as easy as with values

```
{-# language TypeFamilies #-}
```

```
type family MakeOpt (t :: Type) :: Type where  
  MakeOpt (Maybe a) = Maybe a -- already optional  
  MakeOpt (Either e a) = Maybe a -- simplify  
  MakeOpt t           = Maybe t -- other cases
```

```
{-# language GADTs, DataKinds #-}
```

```
data SafeString (escaped :: Bool) where  
  Unsafe  :: String -> SafeString 'False  
  Escaped :: String -> SafeString 'True
```

Type level

Working with types as easy as with values

How to use types to check more invariants

This is **advanced** Haskell

- But we love to talk about it! 💜
- Lots of exploration and research

The *Stitch* paper by Eisenberg is awesome 💎

Many extensions have been stable for *decades*
... yet they won't make into a new Report

GHC2021 is the standard "language version"
since (the recently released) 9.2

Stemming from the **GHC Steering Committee**

- Oversee new extensions to the language
- Similar to processes in other communities

Community

Main players

- GHC Team, builds the compiler
- Haskell Foundation
- Haskell Language Server team
- Several working groups
 - Haskell.org
 - Core Libraries Committee
 - GHC Steering Committee
 - Moving towards Foundation umbrella 

Means of communication

Old school: IRC and mailing lists

- `haskell-cafe`
- `haskell-beginners`


Reddit `/r/haskell`

Quite focused and active

Discourse

Category Theory


A branch of mathematics talking about the abstract structure of things


There's a lot of cross-pollination 

- Functor or monad come from there
- Type theories are influenced by PL research

Category Theory

A branch of mathematics talking about the abstract structure of things

There's a lot of cross-pollination 

However, it's a scary topic for many 

- Not really needed to start with Haskell
- But nevertheless *really* interesting

 It's been a pleasure

Enjoy the rest of Haskell eXchange!

*Let this be the start of a long and exciting
journey in Haskell!*