



Intro to HASKELL for Plutus

About Me

- Started programming in 2005 with C++
- B.S. Electrical Engineering:
 - Minor in Math and Physics
 - Track in Electricity and Magnetism
 - Matlab TA for 3 years
- M.S. Electrical Engineering:
 - Focus in Guidance, Navigation & Control Systems
- Favorite Projects:
 - Building an NFT Launch to Fund Education App (Flutter/Dart, Python)
 - Building an Education App for Android, IOS, and Web (Flutter/Dart, Python, Go?)
 - MTGO Online Trading Bots (C++, SQL)
 - Attitude Determination and Control for a CubeSat (C, Matlab)
 - Auto Pilot & Sensor Integration for Autonomous Vehicles (C, C++, Matlab, Python)
 - Various ML and AI projects

About this Course

- #1 Thing: I'm not an expert in Haskell!
 - Terminology and Best Practices will be a little loose at the beginning
 - Learning Haskell to program contracts in Plutus for Cardano (ADA)
 - Teaching Haskell allows me to help others while mastering the material
- Live Streams on Monday & Wednesday at 7pm Est
- First Hour: Lessons on Haskell Programming
- Second Hour: Solving Challenge Problems in Haskell
 - Typically on CodeWars: <https://www.codewars.com/>

Pre-Reqs (what you Need)

- Free Online Environment:

- <https://replit.com/~>

- Local Environment:

- <https://www.haskell.org/ghc/>
- Windows Install:
 - Directions above
- Linux Install:
 - Performed on Live Stream Day 1
 - <https://www.youtube.com/JBarCode>
- Mac Install:
 - ...when I get my Mini Mac

- Git Repository:

- <https://github.com/JBarCode37/haskell-course>

- Install Git (optional):

- <https://git-scm.com/download>

- Install VS Code (optional)

- <https://code.visualstudio.com/>

How to Support Me (Free to Not-So-Free)

1) Like and Subscribe on YouTube!

- a) <https://www.youtube.com/JBarCode>

2) Follow on Socials:

- a) u/JBarCode on Reddit: <https://www.reddit.com/user/JBarCode>
- b) @JBarCode37 on Twitter: <https://twitter.com/JBarCode37>

3) Buy a Haskell Book with Affiliate Link:

- a) Learn You a Haskell for Great Good by Miran Lipovaca: <https://amzn.to/3c8NiyN>
- b) Programming in Haskell by Graham Hutton: <https://amzn.to/3fVZPCZ>

4) Stake ADA with [KNUGS]

- a) Used to Support an Education App in Development (Launching Sep 2021)
- b) <https://pooltool.io/pool/c5c17e9e1e9fb8044b0215ce9b121f1b8a63723dbfa81c14b7a308ba/epochs>

5) Straight up send me ADA :)

- a) `addr1q8hzsl7hzh164ufhr9hx23cs5wj7hjamye625nmm2474y7w6a2pw5qmntkrpxtt3wcnsjdss7ye5gdmcxn4qhdc6yz9scw48jn`



Last Stream Recap

- We set up a Haskell Programming environment in Linux
- Using Cabal and GHC based on Cardano Node Installation
 - <https://docs.cardano.org/projects/cardano-node/en/latest/getting-started/install.html>
 - <https://www.haskell.org/cabal/download.html>
 - <https://www.haskell.org/ghc/>
- Don't Have that Setup? No Problem! Just use repl.it:
 - <https://replit.com/~>
- gchi
- *.hs haskell script files

Basic Types

- Num:
 - Includes types: Double, Float, Int, Integer, many others
- Float, Double:
 - 0.999, 1.3, -3.2, 0.0, etc.
 - `sqrt 99 :: Float`, `sqrt 99 :: Double`
- Int, Integer:
 - -99, 23, 0, 99, 1, 838383, etc.
 - `2^63, 2^63 :: Int`, `2^63 :: Integer`, `9223372036854775808 :: Int` (this gives a warning)
- Char:
 - 'a', 'c', '\t', '\n', '\8371', etc.
 - Unicode: Try running `putStr ["\t", '\8371', '\n']`

Basic Types Cont.

- String or [Char]:
 - "Hello World", ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'], etc.
- Bool:
 - True, False
- Lists:
 - [1, 2, 3], [True, False, True], ["Hello", "World"], [[0, 0], [0, 4], [5, 0]]
- Tuples:
 - (1, 2, 3), ([1, 2, 3], "Hello World", 99, True, ("Red", False)), yikes!
 - Tuples Create Unique Types (hard to write generic functions for them)

Numeric Operations (...some of them)

- (+): addition
- (-): subtraction, negate
- (*): multiplication
- (^): exponentiation
- (/): division (Fractional values)
- div: division (integer division, Integral values)
 - `div 4 2`, `div 5 2`, `5 `div` 2` (infix notation using backticks)
- mod: remainder from integer division
- abs
- signum

List Operations (...some of them)

- (++): Combine Two List:
 - "Hello " ++ " " ++ "World!"
 - "Hello " ++ name, assumes name is a String
- concat: Combines several items in a list
 - concat ["Hello", " ", "World"]
- reverse:
 - reverse "Hello World", reverse [1, 2, 3, 4, 5]
- head (first item), tail (items after head)
- init (items before last item), last (last item)

Basic Comparison Operation (-> Bool)

- (==): Check for Equality
- (/=): Check for inequality
- (>): Greater Than
- (<): Less Than
- (>=): Greater Than or Equal
- (<=): Less Than or Equal
- :info will give you the priority, but it's often better to use

List Operations (...some more of them)

- `length`: gets the length of a list
- `(!!)`: Retrieve a value (index the list):
 - `"Hello World" !! 3`
- `(:)`: append an item to the front of a list
- `[..]`: Create a ranged list
 - `[1..5]`, `[1, 3..99]`, `[1..]` (use Ctrl-C to interrupt runaway list)
- `(<-)`: List comprehension generator
 - `[n^2 | n <- [1..10]]`
 - There is a lot more to do here. Revisit after logic operations.

Basic Logic Operations Bool -> Bool -> Bool

- (&&): And Operator
 - Only True if both left and right inputs are True
 - Lazy evaluation (if the first input is False, it doesn't check the second input)
- (||): Or Operator
 - Only False if both values are false
 - Lazy evaluation (if the first input is True, it doesn't check the second input)
- not: Inverse (Bool -> Bool)
 - True -> False
 - False -> True

Revisit List Comprehensions

- Logic Checks

- `[x | x <- [1..100], x `mod` 2 == 0]`
- `[x | x <- [1..100], x `mod` 2 == 0, x < 25]`

- Nested List Comprehensions

- `[(x, y, x*y) | x <- [1..10], y <- [1..10]]` -- builds a multiplication table
- ...the sky's the limit here. If we made it this far on day 2, we're probably going too fast