SAN DIEGO FUNCTIONAL PROGRAMMERS

THE BIG ELIXIR

November 7-8, 2019 · New Orleans

https://ti.to/the-big-elixir/the-big-elixir-2019/discount/THEBIGMEETUP2019

Schedule

- 10 min - Intro & Welcome
- 60 min - Intro To Haskell
- 30 min - Kata time
- 20 min - Demo

Intro To Haskell

What Is Haskell

Advanced

Pure

Functional

Declarative

Statically Typed

super impressive words

What Is Haskell

GHCi

```
> ghci
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
Prelude>
Prelude> :?
 Commands available from the prompt:

   <statement>                 evaluate/run <statement>
   :                           repeat last command
   :{\n ..lines.. \n:}\n       multiline command
   :add [*]<module> ...        add module(s) to the current target set
   :browse[!] [[*]<mod>]       display the names defined by module <mod>
                               (!: more details; *: all top-level names)
   :cd <dir>                   change directory to <dir>
   :cmd <expr>                 run the commands returned by <expr>::IO String
   :complete <dom> [<rng>] <s> list completions for partial input string
   :ctags[!] [<file>]          create tags file <file> for Vi (default: "tags")
                               (!: use regex instead of line number)
   :def <cmd> <expr>           define command :<cmd> (later defined command has
                               precedence, ::<cmd> is always a builtin command)
   :doc <name>                 display docs for the given name (experimental)
   :edit <file>                edit file
   :edit                       edit last module
   :etags [<file>]             create tags file <file> for Emacs (default: "TAGS")
   :help, :?                   display this list of commands
   :info[!] [<name> ...]       display information about the given names
```

```
Prelude> 1 + 1
2
Prelude> 3 - 2
1
Prelude> [1] ++ [2]
[1,2]
Prelude> "Hello, " ++ "World!"
"Hello, World!"
Prelude>
Prelude> :t (+)
(+) :: Num a ⇒ a → a → a
Prelude>
Prelude> :l hello.hs
[1 of 1] Compiling Main             ( hello.hs, interpreted )
Ok, one module loaded.
```

```
*Main> hello "Jesse"
"Hello,Jesse"
*Main>
```

```
*Main> :info hello
hello :: [Char] -> [Char]   -- Defined at hello.hs:1:1
*Main>
```

Functions

```
hello name = "Hello, " ++ name
hello “SDFP”
hello name = "Hello, " ++ name
```

```haskell
hello :: [Char] → [Char]
hello name = "Hello, " ++ name
hello :: String → String
hello name = "Hello, " ++ name
```

```haskell
add :: Num a ⇒ a → a → a
add x y = x + y
```

Currying

```haskell
add :: Num a ⇒ a → a → a
add x y = x + y
```

```haskell
add:: Num a => a -> (a -> a)
add x y = x + y
```

```haskell
add:: Num a => a -> (a -> a)
add x y = x + y
```

Types

Unary Types

```
data SDFP = SDFP
```

Type Aliases

```
type Goats = Int
```

newtype

```
newtype Goats = Goats Int
```

```haskell
newtype Goats = Goats Int
```

```haskell
newtype Cows = Cows Int
```

```haskell
tooManyGoats :: Goats → Bool
tooManyGoats (Goats n) = n > 42
```

Algebraic Data Types

Sum Types

```haskell
data Bool = True | False
data Foo = Bar | Baz | Qux
data WhatIsThis = String | Integer
```

Product Types

```
data FirstLast = FirstLast String String
```

```
type FirstLastTuple = (String, String)
```

Records

```haskell
data FirstLast =
  FirstLast { first :: String,
            , last :: String }
```

Type Arguments

```haskell
data Perhaps a = Nope
               | Yessir a
```

```haskell
data Or a b = This a
            | That b
```

Pattern Matching

```
data Perhaps a = Nope
               | Yessir a
```

```haskell
mappity :: Perhaps a → (a → b) → Perhaps b
mappity Nope _ = Nope
mappity (Yessir a) f = Yessir (f a)
```

```haskell
data Listy a = Nil
             | Cons a (Listy a)
```

```haskell
addListy :: Num a ⇒ Listy a → a
addListy Nil = 0
addListy (Cons n listy) = n + addListy listy
     addListy :: Num a ⇒ Listy a → a
     addListy Nil = 0
```

Type Classes

```
Prelude> :info Num
class Num a where
  (+) :: a → a → a
  (-) :: a → a → a
  (*) :: a → a → a
  negate :: a → a
  abs :: a → a
  signum :: a → a
  fromInteger :: Integer → a
  {-# MINIMAL (+), (*), abs, signum, fromInteger, (negate | (-)) #-}
    -- Defined in ‘GHC.Num’
instance Num Word -- Defined in ‘GHC.Num’
instance Num Integer -- Defined in ‘GHC.Num’
instance Num Int -- Defined in ‘GHC.Num’
instance Num Float -- Defined in ‘GHC.Float’
instance Num Double -- Defined in ‘GHC.Float’
```

```
Prelude> :i Eq
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
  {-# MINIMAL (==) | (/=) #-}
    -- Defined in ‘GHC.Classes’
```

Type Class Deriving

```haskell
data Listy a = Nil
             | Cons a (Listy a)
```

```haskell
data Listy a = Nil
             | Cons a (Listy a)
             deriving (Show, Eq)
```

```
Prelude> :load hello.hs
[1 of 1] Compiling Hello            ( hello.hs, interpreted )
Ok, one module loaded.
*Hello> list1 = Nil
*Hello> list2 = Cons 2 (Cons 1 Nil)
*Hello> list1 == list2
False
*Hello> list1
Nil
*Hello> list2
Cons 2 (Cons 1 Nil)
```

Type Class Instances

```haskell
data Listy a = Nil
             | Cons a (Listy a)
             deriving (Eq)
```

```haskell
instance Show a => Show (Listy a) where
   show Nil = "[ ]”
   show (Cons a listy) = "[ " ++ show a ++ ", " ++ show listy ++ " ]"
```

```
*Hello> list1
[ ]
*Hello> list2
[ 2, [ 1, [ ] ] ]
```

Tools

HaskellLanguage: https://www.haskell.org

Hoogle: https://hoogle.haskell.org

Hackage: https://hackage.haskell.org

Stackage: https://www.stackage.org

GHCID: https://github.com/ndmitchell/ghcid

Haskelly: https://marketplace.visualstudio.com/items?itemName=UCL.haskelly

Haskero: https://marketplace.visualstudio.com/items?itemName=Vans.haskero

Intero: https://github.com/chrisdone/intero

Stackage: https://www.stackage.org

References

Haskell Programming: http://haskellbook.com/

The Joy of Haskell: https://joyofhaskell.com

Type Classes: https://typeclasses.com/

Typeclassopedia: https://wiki.haskell.org/Typeclassopedia

Haskell Programming: http://haskellbook.com/

The Joy of Haskell: https://joyofhaskell.com

Type Classes: https://typeclasses.com/

Typeclassopedia: https://wiki.haskell.org/Typeclassopedia

CIS 194: https://www.seas.upenn.edu/~cis194/spring13/lectures.html

A Type of Programming: https://atypeofprogramming.com/

The End

https://bit.ly/sdfp-aug-2019

Some Guidelines

- Try to work in groups!
- Prefer recursion to loops
- No mutation
- If all else fails, forget the guidelines