COMP3258
Functional Programming

Tutorial Session 1: Introduction to Haskell Development

About Me

- I am a tutor of FP 2023.
- I was a tutor of FP 2022.
  - My responsibility is to help your understand FP (Haskell) better.
  - My job will cover design and grade of assignments, but blind to exams.
- I am a Haskell enthusiast, and happy to answer any related questions.
- I am a PhD student of Bruno and do research about FP.

How to reach (1)

- Course Instructor (Bruno C. d. S. **Oliveira**) bruno@cs.hku.hk
- Tutor (Xu **Xue**) xxue@cs.hku.hk

Blue font is *last name*.

How to reach (II)

- Please send your inquiries to the instructor's or tutor's email box with the subject beginning with *COMP3258*.
- You are encouraged to post questions on the discussion forum.
- If you want to meet outside office hours, please send us an email to arrange a meeting.
  - Emails are guaranteed to reply within 3 working days.

About Tutorial

- It's optional but encouraged to attend.
- The structure of the session is
  - Review (20 mins)
  - Code Practice (30 mins)

Haskell (haskell.org)

- GHC, GHCi and GHCup
- Hoogle
- Cabal, Stack

> ghcup tui

```
⌥⌘1   ghci (ghc-9.6.2)   ⌘1   ghcup (ghcup)   ⌘2

                                GHCup
    Tool  Version        Tags                        Notes
✔✔  GHCup 0.1.19.4       latest,recommended
✗                        latest
✗   Stack 2.9.3          recommended
✗   Stack 2.9.1
✔✔  Stack 2.7.5
✓   HLS   2.2.0.0        latest,recommended
✔✔  HLS   2.1.0.0
✗   HLS   2.0.0.1
✓   HLS   2.0.0.0
✗   HLS   1.10.0.0
✗   HLS   1.9.1.0
✓   HLS   1.8.0.0
✓   HLS   1.7.0.0
✗                        latest
✗   cabal 3.8.1.0
✔✔  cabal 3.6.2.0        recommended
✔✔  GHC   9.6.2          latest,base-4.18.0.0        hls-powered

q:Quit  i:Install  u:Uninstall  s:Set  c:ChangeLog  a:Show all versions  t:Show all tools  ↑:Up  ↓:Down
```

hoogle.haskell.org

https://www.cis.u... | beautiful concurr... | https://blogs.asar... | hoogle.haskell.org

Manual | haskell.org

Hoogle

Num a => a -> a -> a | set:stackage | Search

Packages

- is:exact
- base
- hedgehog
- ghc
- base-compat
- protolude
- relude
- rio
- base-prelude
- classy-prelude
- basic-prelude
- universum
- Cabal-syntax
- github
- ghc-lib-parser
- prelude-compat
- rebase
- opaleye
- xmonad-contrib
- stack
- incipit-base
- LambdaHack
- cabal-install-solver
- dimensional
- mixed-types-num
- linear-base
- base-unicode-symbol

:: Num a => a -> a -> a

(+) :: Num a => a -> a -> a

base Prelude GHC.Num, hedgehog Hedgehog.Internal.Prelude, ghc GHC.Prelude.Basic, base-compat Prelude.Compat, protolude Protolude Protolude.Base, relude Relude.Numeric, rio RIO.Prelude, base-prelude BasePrelude BasePrelude.Operators, classy-prelude ClassyPrelude, basic-prelude CorePrelude, universum Universum.Base, Cabal-syntax Distribution.Compat.Prelude, github GitHub.Internal.Prelude, ghc-lib-parser GHC.Prelude.Basic, prelude-compat Prelude2010, rebase Rebase.Prelude, opaleye Opaleye.Operators, xmonad-contrib XMonad.Config.Prime, stack Stack.Prelude, incipit-base Incipit.Base, LambdaHack Game.LambdaHack.Core.Prelude, cabal-install-solver Distribution.Solver.Compat.Prelude

(-) :: Num a => a -> a -> a

base Prelude GHC.Num, hedgehog Hedgehog.Internal.Prelude, ghc GHC.Prelude.Basic, base-compat Prelude.Compat, protolude Protolude Protolude.Base, relude Relude.Numeric, rio RIO.Prelude, base-prelude BasePrelude BasePrelude.Operators, classy-prelude ClassyPrelude, basic-prelude CorePrelude, universum Universum.Base, Cabal-syntax Distribution.Compat.Prelude, github GitHub.Internal.Prelude, ghc-lib-parser GHC.Prelude.Basic, prelude-compat Prelude2010, rebase Rebase.Prelude, opaleye Opaleye.Operators, xmonad-contrib XMonad.Config.Prime, stack Stack.Prelude, incipit-base Incipit.Base, LambdaHack Game.LambdaHack.Core.Prelude, cabal-install-solver Distribution.Solver.Compat.Prelude

(*) :: Num a => a -> a -> a

base Prelude GHC.Num, hedgehog Hedgehog.Internal.Prelude, ghc GHC.Prelude.Basic, base-compat Prelude.Compat, protolude Protolude Protolude.Base, relude Relude.Numeric, rio RIO.Prelude, base-prelude BasePrelude BasePrelude.Operators, classy-prelude ClassyPrelude, basic-prelude CorePrelude, universum Universum.Base, Cabal-syntax Distribution.Compat.Prelude, github GitHub.Internal.Prelude, ghc-lib-parser GHC.Prelude.Basic, prelude-compat Prelude2010, rebase Rebase.Prelude, opaleye Opaleye.Operators, xmonad-contrib XMonad.Config.Prime, stack Stack.Prelude, incipit-base Incipit.Base, LambdaHack Game.LambdaHack.Core.Prelude

subtract :: Num a => a -> a -> a

base Prelude GHC.Num, hedgehog Hedgehog.Internal.Prelude, ghc GHC.Prelude.Basic, base-compat Prelude.Compat, protolude Protolude Protolude.Base, relude Relude.Numeric, rio RIO.Prelude, base-prelude BasePrelude, classy-prelude ClassyPrelude, basic-prelude CorePrelude, universum Universum.Base, Cabal-syntax Distribution.Compat.Prelude, github GitHub.Internal.Prelude, ghc-lib-parser GHC.Prelude.Basic, prelude-compat Prelude2010, dimensional Numeric.Units.Dimensional.Prelude, rebase Rebase.Prelude, mixed-types-num Numeric.MixedTypes.PreludeHiding, xmonad-contrib XMonad.Config.Prime, stack Stack.Prelude, linear-base Prelude.Linear, incipit-base Incipit.Base, LambdaHack Game.LambdaHack.Core.Prelude

the same as `flip (-)`. Because – is treated specially in the Haskell grammar, `(– e)` is not a section, but an application of prefix negation. However, `(subtract exp)` is equivalent to the disallowed section.

(–) :: Num a => a -> a -> a

base-unicode-symbols Prelude.Unicode

a – b = a - b U+2212, MINUS SIGN

Cabal

EXPLORER

CODE
- app
  - Main.hs
- dist-newstyle
- CHANGELOG.md
- Code.cabal

OUTLINE

TIMELINE

Code.cabal

```
cabal-version:      2.4
name:               Code
version:            0.1.0.0

-- A short (one-line) description of the package.
-- synopsis:

-- A longer description of the package.
-- description:

-- A URL where users can report bugs.
-- bug-reports:

-- The license under which the package is released.
-- license:
author:             Xu Xue
maintainer:         juniorxxue@gmail.com

-- A copyright notice.
-- copyright:
-- category:
extra-source-files: CHANGELOG.md

executable Code
    main-is:          Main.hs

    -- Modules included in this executable, other than Main.
    -- other-modules:
```

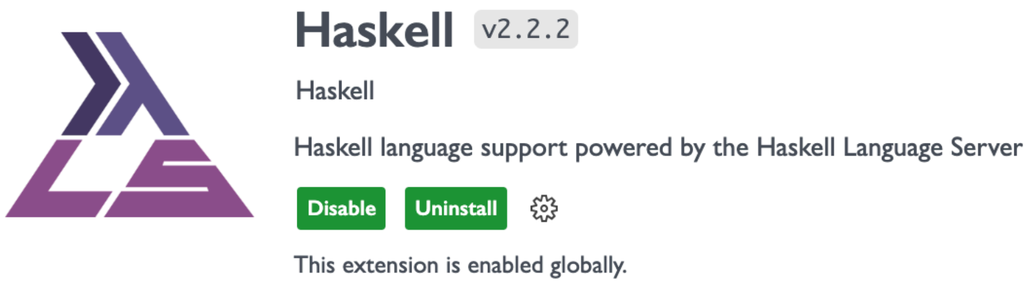
Some useful commands:

1. cabal init
2. cabal install
3. cabal run

0 0 Ln 10, Col 1 Spaces: 4 UTF-8 LF Cabal Spell

Editor

- VSCode (Haskell Plugin)
  - Follow the instructions to install the HLS
- Emacs/Vim/Sublime… (with LSP)
  - Not recommended



Editor Feat. 1: Type Check

```
94  bar :: Bool
95  bar = 1
96
```

```
_ :: Bool

• No instance for ‘Num Bool’ arising from the literal ‘1’
• In the expression: 1
  In an equation for ‘bar’: bar = 1 typecheck(-Wdeferred-type-errors)
```

View Problem (⌥F8) Quick Fix... (⌘.)

Editor Feat. 2: Type Inference

```
92
93
    f :: Integer -> Integer -> Integer
94  f = \x y -> x + y
95
96
```

However, In this course, you are encouraged to write the type first.

Editor Feat. 3: Code Format

```
f :: Integer -> Integer -> Integer
94  f = \x y -> x + y
95
96
```

```
Redundant lambda
Found:
  f = \ x y -> x + y
Why not:
  f x y = x + y
 hlint(refact:Redundant lambda)
```

Learn from the hint.

Editor Feat.4: Quick Evaluation

```
93
   Evaluate...
94 -- >>> 1 + 2
```

```
93
   Refresh...
94 -- >>> 1 + 2
95 -- 3
```

Editor Feat.5: Docs + Type Instantiation

```
sum123 :: Int
sum123 = foldr (+) 0 [1,2,3]
```

```
foldr :: forall (t :: Type → Type) a b.
Foldable t ⇒
(a → b → b) → b → t a → b
```

Defined in 'Data.Foldable' (base-4.18.0.0)

Right-associative fold of a structure, lazy in the accumulator.

In the case of lists, `foldr`, when applied to a binary operator, a starting value (typically the right-identity of the operator), and a list, reduces the list using the binary operator, from right to left:

```
foldr f z [x1, x2, ..., xn] = x1 `f` (x2 `f` ... (xn `f` z)...)
```

Note that since the head of the resulting expression is produced by an application of the operator to the first element of the list,

```
sum123 :: Int
sum123 = foldr (+) 0 [1,2,3]
```

```
[1,4,7,10,13]
```

Documentation

Source

```
_ :: (Int → Int → Int) → Int → [Int] → Int
_ :: forall a b. (a → b → b) → b → [a] → b
_ :: forall (t :: Type → Type) a b.
Foldable t ⇒
(a → b → b) → b → t a → b
```

"foldr": Unknown word. eSpell

Dive into Haskell

- by REPL
- by Script

GHCi (The REPL)

```
ghci> 2 + 3 * 4
14
ghci> (2 + 3) * 4
20
ghci> sqrt (3^2 + 4^2)
5.0
ghci> 1 == 2
False
```

Some Useful Functions

```
ghci> :t head
head :: GHC.Stack.Types.HasCallStack ⇒ [a] → a
ghci> head [1,2,3,4]
1

ghci> :t tail
tail :: GHC.Stack.Types.HasCallStack ⇒ [a] → [a]
ghci> tail [1,2,3,4]
[2,3,4]

ghci> :t (!!)
(!!) :: GHC.Stack.Types.HasCallStack ⇒ [a] → Int → a
ghci> [1,2,3,4] !! 2
3

ghci> :t take
take :: Int → [a] → [a]
ghci> take 3 [1,2,3,4]
[1,2,3]

ghci> :t drop
drop :: Int → [a] → [a]
ghci> drop 3 [1,2,3,4]
[4]
```

which is called "indexing"

```
ghci> :t reverse
reverse :: [a] → [a]
ghci> reverse [1,2,3,4]
[4,3,2,1]

ghci> :t length
length :: Foldable t ⇒ t a → Int
ghci> length [1,2,3,4]
4

ghci> :t (++)
(++) :: [a] → [a] → [a]
ghci> [1, 2, 3] ++ [4, 5]
[1,2,3,4,5]
```

which is called "concatenation"

GHCi: Script

1. Create Test.hs
2. First line write down: module Test where
3. Then write your definitions
4. Use GHCi to load (:l Test.hs) and reload (:r)

Main.hs | Test.hs

```haskell
module Test where

double :: Int -> Int
double x = x + x

quadruple :: Int -> Int
quadruple x = double (double x)

-- then we add more functions
factorial :: Int -> Int
factorial n = product [1..n]

average :: [Int] -> Int
average ns = sum ns `div` length ns
```

PROBLEMS 5 | OUTPUT | DEBUG CONSOLE | TERMINAL

```
λ ~/Library/CloudStorage/Dropbox/cs/2023comp3258/tutorial/01/Code/app/ ghci
GHCi, version 9.6.2: https://www.haskell.org/ghc/  :? for help
ghci> :l Test.hs
[1 of 1] Compiling Test             ( Test.hs, interpreted )
Ok, one module loaded.
ghci> double 2
4
ghci> quadruple 4
16
ghci> :r
Ok, one module loaded.
ghci> factorial 10
3628800
ghci> average [1,2,3,4,5]
3
ghci>
```

Prelude

Main.hs

```
1  module Main where
2
3  x :: Integer
4  x = 1
5
6  main
7  main
```

imported by Prelude!

```
Integer :: Type
```

Defined in 'GHC.Num.Integer' (ghc-bignum-1.3)

Arbitrary precision integers. In contrast with fixed-size integral types such as `Int`, the `Integer` type represents the entire infinite range of integers.

Integers are stored in a kind of sign-magnitude form, hence do not expect two's complement form when using bit operations.

If the value is small (fit into an `Int`), `IS` constructor is used. Otherwise `IP` and `IN` constructors are used to store a `BigNat` representing respectively the positive or the negative value magnitude.

Prelude

- Prelude is a module that contains a small set of standard definitions.
- It's imported by default implicitly.
- What Prelude does is only to re-import definitions in other modules.
- Check its source code at https://hackage.haskell.org/package/base-4.18.0.0/docs/src/Prelude.html

Questions

Q1

- Is the following Haskell program valid?

```
N = a `div` length xs where
            a  = 10
            xs = [1,2,3,4,5]
```

- Surprisingly the program is invalid. The name of a usual function (or value) definition shouldn't start with a capital letter. If you feed the program to GHCi, it would emit an error message:

```
Test.hs:16:1: error: [GHC-76037] Not in scope: data constructor ‘N’
   |
16 | N = a `div` length xs where
   | ^
Failed, no modules loaded.
```

Q2

- Define your own last function using the functions introduced above (head, tail, take, drop, reverse, length)?
- Hint: last returns the last element of a list.
- Are there other approaches?

```haskell
-- If we are restricted to use library function only
last :: [a] → a
last xs = head (drop (length xs - 1) xs)

-- While if we implement it by ourself, we could do it by pattern matching
last :: [a] → a
last [] = error "calling last on empty list"
last (x : []) = x
last (x : y : xs) = last (y : xs)
```

Q3

- Define init in two different ways.
- Hint: init removes the last element of a list.

```haskell
-- using library functions
init :: [a] → [a]
init xs = take (length xs - 1) xs

-- pattern matching and recursion
init :: [a] → [a]
init [] = error "calling init on empty list"
init (x : []) = []
init (x : y : xs) = x : init (y : xs)
```

Further Reading

- *Programming in Haskell* (Graham Hutton)
- *Learn You a Haskell for Great Good!* (Miran Lipovača)
- *Haskell Programming from First Principles* (Christopher Allen, Julie Moronuki)

Q & A