Haskell
Lecture 6

Higher-Order Functions

Functions are Data

- *Combine using operators*
    - *f . g -- composition for example*
- *lambda expressions, describe function by expression*
- *can be inputs, outputs of other functions*
- *can be partially applied*

Function Composition

- *(f . g) x has same meaning as f (g x)*
- *The above means apply g to x, then apply f to result*
- *The application is in the reverse of the listed order*
- *operator >.> can be defined as reverse order to .*
    - *let f >.> g = g . f*

Application operator $

- *Given function f,*
  - *f e -- apply function f to argument e*
  - *f $ e -- apply function f to argument e*
  - *If g is actually a function as in f (g e), $ is used to remove parentheses, as in f $ g e*

Lambda Abstractions

- Write a function directly without giving it a name
- Syntax   \v -> expression
- From lambda calculus, Haskell Curry was one of the inventors
- \ is close to the Greek letter lambda.
- lamda.hs has examples

Partial Application

- *Given a function in 2 arguments, when applied to 1 argument yields a function in 1 argument*
- *This can be extended to functions of 3 or more arguments with fewer arguments supplied to it.*
- *See partial.hs for examples*

Curried Functions

- *Functions in Haskell are represented in curried form*
  - *curry named after Haskell Curry*
  - *curried form: functions take arguments one at a time*
  - *add :: Int -> Int -> Int is actually short for*
    add :: Int -> (Int -> Int)
- *This is why partial applications work*

Uncurried Functions

- *Normally functions are curried, define by*
    - $f\ x\ y = x + y$
- *For uncurried, do*
    - $f\ (x,y) = x + y$
- *The arguments are grouped into a tuple*

Operator Sections

- *Partially applied operator defined functions*
- *(op x) y means y op x*
- *(x op) y means x op y*
    - *(-2) x is x - 2*
    - *(2-) x is 2 - x*

return with IO type

- *return is not the same in Haskell as in C (or Java)*
- *return does not mean leave called function to go back to calling function (with or without value)*
- *return means wrap a value in IO*
- *See return.hs*