HASKELL BASICS AND TYPES

Curt Clifton
Rose-Hulman Institute of Technology

Check out `HaskellInClass` from SVN

MORE BASICS

HASKELL ERROR MESSAGES

- Could they be more cryptic?
- Example: *haar xs = haarLevelN (logBase 2 (length xs)) xs*

```
No instance for (Floating Int)
  arising from a use of `logBase' at functions.hs:31:22-42
Possible fix: add an instance declaration for (Floating Int)
In the first argument of `haarLevelN', namely
    `(logBase 2 (length xs))'
In the expression: haarLevelN (logBase 2 (length xs)) xs
In the definition of `haar':
    haar xs = haarLevelN (logBase 2 (length xs)) xs
```

HASKELL ERROR MESSAGES

- *:type logBase → logBase :: (Floating a) => a -> a -> a*
- Read: "given that *a* is a Floating point type, then *logBase* is a function that takes two arguments of type *a* and returns a value of type *a*"
- Learning to read Haskell errors will take time
- Solution to the current problem:
  haar xs = haarLevelN (logBase 2 (fromIntegral (length xs))) xs

HASKELL IS: LAZY

- No computation takes place unless it is forced to when the result is used
- Let's us make infinite lists!
  - Example: *makeList = 1 : makeList*
- Useful function from the Prelude: *take n xs*
- Try writing: *upFrom n*
  - Example: *upFrom 5* yields *[5,6,7,8,…]*

Q1,2

HASKELL IS: CASE SENSITIVE

- Functions **must** start with lower case
- Types **must** start with upper case
  - More info. on types coming…

HASKELL IS: Mostly PURELY FUNCTIONAL

- Given the same arguments, a function in Haskell always produces the same results
- Sometimes referred to as *referential transparency*
- This allows automatic *memoization*
  - Storing the results of previously evaluated functions
- Mostly? Impurity needed for I/O and persistence

HASKELL IS: STRONGLY, STATICALLY TYPED

- All types must be given or inferable (guess-able) at "compile" time
- *Type inference*: known (or inferred) types of functions and arguments are used to infer types of other arguments and functions
- Try this: *:t 11 + 31*
  :t 11 + 31::Int
  :t (+)

‘IF’ IS AN EXPRESSION

- *myDrop n xs = if n <= 0 || null xs*
  then xs
  else myDrop (n - 1) (tail xs)
- Can’t have a one-legged *if* in Haskell. Why not?

FUN WITH LISTS

- What is the type of *map*, *filter*, *foldr*, *foldl*, *zip*, *zipWith*?
- Try:
  - Add *import List* to top of your *basics.hs* file
  - Reload, then enter
    - *:browse List*
    - *:info filter*
- Recall: *[1..10]* yields *[1,2,3,4,5,6,7,8,9,10]*

Also see *http:// www.haskell.org / ghc / docs / latest / html / libraries /*

LAZY FIB

Gives the n^{th} element of *fibList*

- *fastFib n = fibList !! n*
  where fibList = 0 : 1 : zipWith (+) fibList (tail fibList)

Parentheses turn infix operator into a function

TYPES

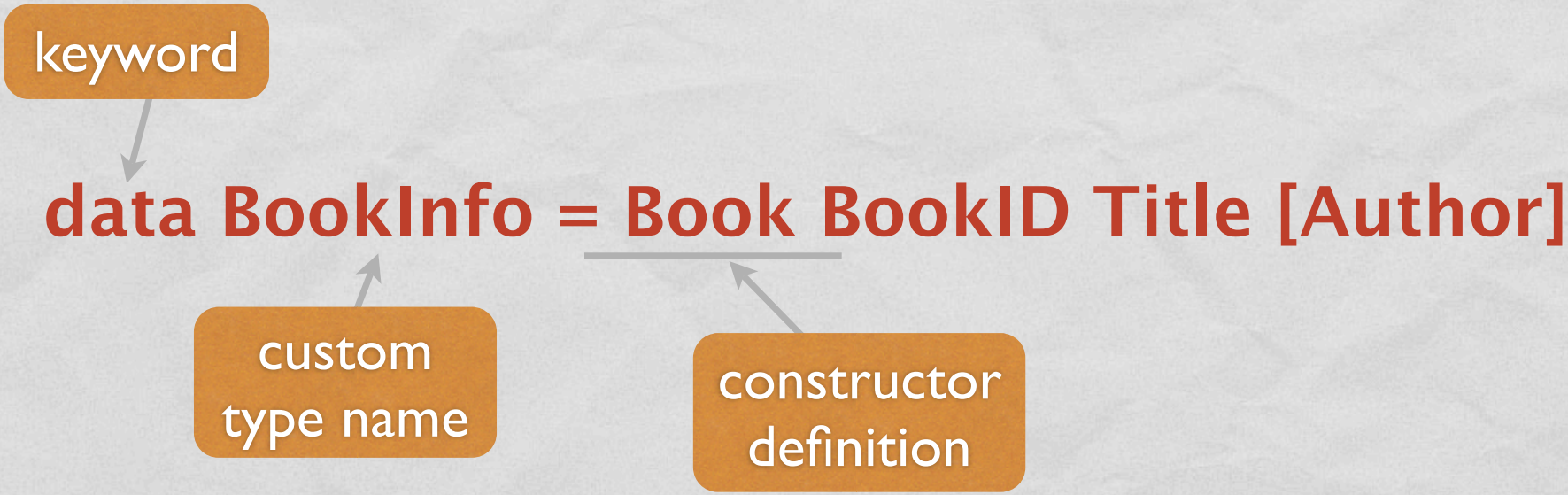
DECLARING TYPES OF FUNCTIONS

- We can declare specific types for functions:
  - *upFrom :: (Num a) => a -> [a]*
- Why useful?
- Helpful hint for learning types:
  - Make ghci display the type of each result by entering: *:set +t*
  - Add it to *ghci.conf* if you want

TYPE SYNONYMS

- Type synonyms let us give additional names to existing types → improves readability
- *type BookID = Int*
  type Title = String
  type Author = String

DECLARING CUSTOM DATA TYPES



Try: *:t Book*

:t Book 123 "Little Schemer" ["Friedman","Felleisen"]

CUSTOM DATA TYPES

- Use constructors to make values with the custom type

```
>>> Book 123 "ls" ["f","f"]
```

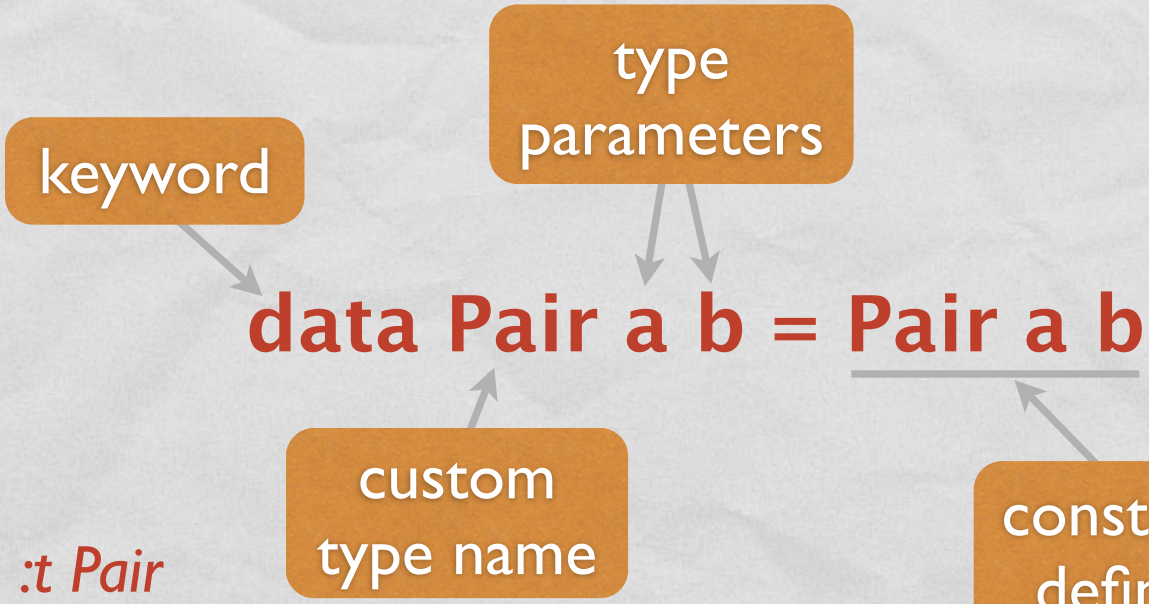
- Can make custom types *instances* of *type classes*

```
data BookInfo = Book …
  deriving (Show)
```

- Can pattern match against the types

```
title (Book _ t _) = t
firstAuth (Book _ _ (x:_)) = x
```

POLYMORPHIC CUSTOM DATA TYPES




```
*Main> :t Pair
Pair :: a -> b -> Pair a b
*Main> :t Pair 'c' "Saw"
Pair 'c' "Saw" :: Pair Char [Char]
*Main> :t Pair 1 'c'
Pair 2 'c' :: (Num t) => Pair t Char
```

type name and constructor name can be the same

CONSIDER…

- Consider:

```
findElement :: (a -> Bool) -> [a] -> a
findElement p (x:xs) =
    if p x
    then x
    else findElement p xs
```

- What should we do if we don't find a match?

MULTIPLE CONSTRUCTORS AND THE *MAYBE* TYPE

- The Haskell Prelude defines a custom type:
  - *data Maybe a = Nothing*
    | Just a
- Example:
  - ```
    findElement2 :: (a -> Bool) -> [a] -> Maybe a
    findElement2 _ [] = Nothing
    findElement2 p (x:xs) =
        if p x
        then Just x
        else findElement2 p xs
    ```
    ```