Using Types

Slides thanks to Mark Jones
Expressions Have Types:

• The *type* of an expression tells you what kind of value you might expect to see if you evaluate that expression.

• In Haskell, read “::” as “has type”.

• Examples:
  - 1 :: Int, 'a' :: Char, True :: Bool, 1.2 :: Float, ...

• You can even ask GHCI for the type of an expression: :t expr.
Type Errors:

Prelude> 'a' && True

<interactive>:26:1:
  Couldn't match expected type `Bool' with actual type `Char'
  In the first argument of `(&&)', namely 'a'
  In the expression: 'a' && True
  In an equation for `it': it = 'a' && True

Prelude> odd 1 + 2

<interactive>:29:7:
  No instance for (Num Bool)
  arising from a use of `+
  Possible fix: add an instance declaration for (Num Bool)
  In the expression: odd 1 + 2
  In an equation for `it': it = odd 1 + 2
Pairs:

• A pair packages two values into one
  
  (1, 2)   ('a', 'z') (True, False)

• Components can have different types
  
  (1, 'z')   ('a', False)   (True, 2)

• The type of a pair whose first component is of type A and second component is of type B is written (A,B)
• What are the types of the pairs above?
Operating on Pairs:

• There are built-in functions for extracting the first and second component of a pair:
  – \( \text{fst} \ (\text{True}, \ 2) = \text{True} \)
  – \( \text{snd} \ (0, \ 7) = 7 \)

• Is the following property true?

  For any pair \( p \), \( (\text{fst} \ p, \ \text{snd} \ p) = p \)
Lists:

• Lists can be used to store zero or more elements, in sequence, in a single value:
  
  
  [ ]  [1, 2, 3]  ['a', 'z']  [True, True, False]

• All of the elements in a list must have the same type

• The type of a list whose elements are of type A is written as [A]

• What are the types of the lists above?
Overloading

- Some expressions can have more than one type

- Examples
  - 23
  - []
  - \( f \ x = x < 3 \)
  - \( f \ x = \text{show} \ x ++ \text{“ is the answer”} \)
One way to get these is overloading

• Three important causes of overloading

• Numbers
  – Num

• Comparisons
  – Ord

• Displaying as a string
  – Show
Information about overloading

• By typing “:i T“ to GHCI you can find out details of about the “T” kind of overloading.

• For example

• :i Show
• :i Num
Example: Num

*ProgrammingOutLoud> :i Num

```haskell
class (Eq a, Show a) => Num a where
    (+) :: a -> a -> a
    (*) :: a -> a -> a
    (-) :: a -> a -> a
    negate :: a -> a
    abs :: a -> a
    signum :: a -> a
    fromInteger :: Integer -> a
```

```haskell
    -- Defined in GHC.Num
instance Num Int -- Defined in GHC.Num
instance Num Integer -- Defined in GHC.Num
instance Num Double -- Defined in GHC.Float
instance Num Float -- Defined in GHC.Float
```
Integer

• Constants like 5, 35, 897 are in the Num class

• They default to the type Integer
Double

• Constants like 5.6, and 0.0 are Fractional

• These default to the type Double
Type declarations

• If you have a problem with a numeric constant like 5 or 78.9, you will probably see an error that mentions Num or Fractional.

• Fix these by adding type declarations