List Processing

Composite Data Types in Haskell

Prof Hans Georg Schaathun

Høgskolen i Ålesund

February 2, 2015

hials

Outline

hials

Polymorphism

- Consider the function `length`
  - it returns the number of elements in a list
- Many functions?
  - `length :: [Integer] -> Integer`
  - `length :: [Double] -> Integer`
  - `length :: [Bool] -> Integer`
  - Do we have to make our own for `[Customer]`?
- No, we have polymorphism
  - `length :: [a] -> a`
  - `a` is an arbitrary type

hials

Polymorphism

- Consider the function `length`
  - it returns the number of elements in a list
- Many functions?
  - `length :: [Integer] -> Integer`
  - `length :: [Double] -> Integer`
  - `length :: [Bool] -> Integer`
  - Do we have to make our own for `[Customer]`?
- No, we have polymorphism
  - `length :: [a] -> a`
  - `a` is an arbitrary type

hials

Polymorphism

- Consider the function `length`
  - it returns the number of elements in a list
- Many functions?
  - `length :: [Integer] -> Integer`
  - `length :: [Double] -> Integer`
  - `length :: [Bool] -> Integer`
  - Do we have to make our own for `[Customer]`?
- No, we have polymorphism
  - `length :: [a] -> a`
  - `a` is an arbitrary type

hials

Another example

1. `zip :: [a] -> [b] -> [(a,b)]`
2. Two arbitrary types `a` and `b`
3. `zip ['a'..'f'] [0..]`
4. gives
   `[('a',0),('b',1),('c',2),('d',3),('e',4),('f',5)]`

hials

Overloading

- Polymorphic functions
  1. one definition for multiple types
  2. e.g. `fst (x,y) = x`
- Overloading
  1. one function name for different definitions
  2. different definitions for different types
  3. e.g. `x == y`
- We will get back to overloading later

hials

Outline

hials

Some list functions

`!!`	`[a] -> Int -> a`	get element by index
`head last`	`[a] -> a`	get first/last element
`tail init`	`[a] -> Int -> [a]`	get all elements but the first/last
`reverse`	`[a] -> [a]`	reverse order
`replicate`	`Int -> a -> [a]`	a list repeating the same element

hials

Standard libraries

1. Prelude is loaded by default
2. Other standard modules can be loaded
   1. Hierarchical structure with dotted notation
   1. Data.List
   2. Data.Array
3. Third party libraries from HackageDB
   - package manangement with cabal

hials

The Haskell Platform

1. Compiler and interpreter: GHC/GHCi
2. Standard libraries
3. Package management tool: cabal
4. C/Haskell interface tool: Hsc2hs
5. Other tools

hials