

Haskell: Programming with Functions

Overview of Day 2

Niki Vazou, Pablo Serrano
Clubes de Ciencia, Summer 2015

Cabal: Haskell Package Manager

Package or Library:

a group of functions that implement a functionality

Data.Char

Data.List

Prelude

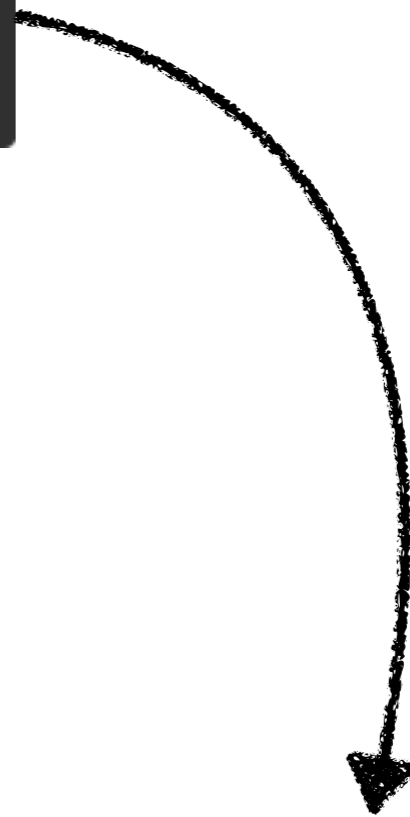
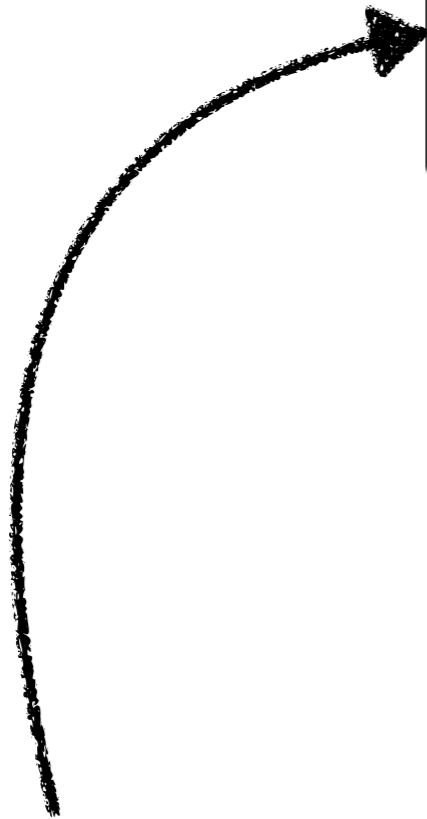
NLP.Stemmer

Cabal: Haskell Package Manager

A cabal is a group of people united in some close design together, usually to promote their private views or interests in a church, state, or other community, often by intrigue, usually unbeknownst to persons outside their group.*

*Wikipedia

Cabal: Haskell Package Manager



Haskell Developer

Haskell User

Cabal: Haskell Package Manager



Download: ``cabal install steemer``

import in code: ``import NPL.Steemer``

Importing things

```
import NPL.Steemer
```

```
import Data.Char (toLower)
```

```
import Prelude hiding (tail)
```

```
import qualified Data.Char as C
```

Type Classes

A type class is an interface that defines some behavior.

class name

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
```

class method(s)

Type Class Instances

Define how a type instantiates a class

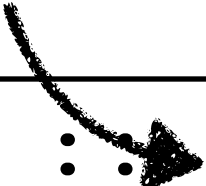
```
instance Eq a => Eq (Err a) where  
  err1 == err2 = eqErr err1 err2
```

Now I can use (==) on `Err a` expressions

Type Class Constraints

Constrain type variables to be instances of type class

constraint



```
(==) :: (Eq a) => a -> a -> Bool
```

Constraints are propagated!

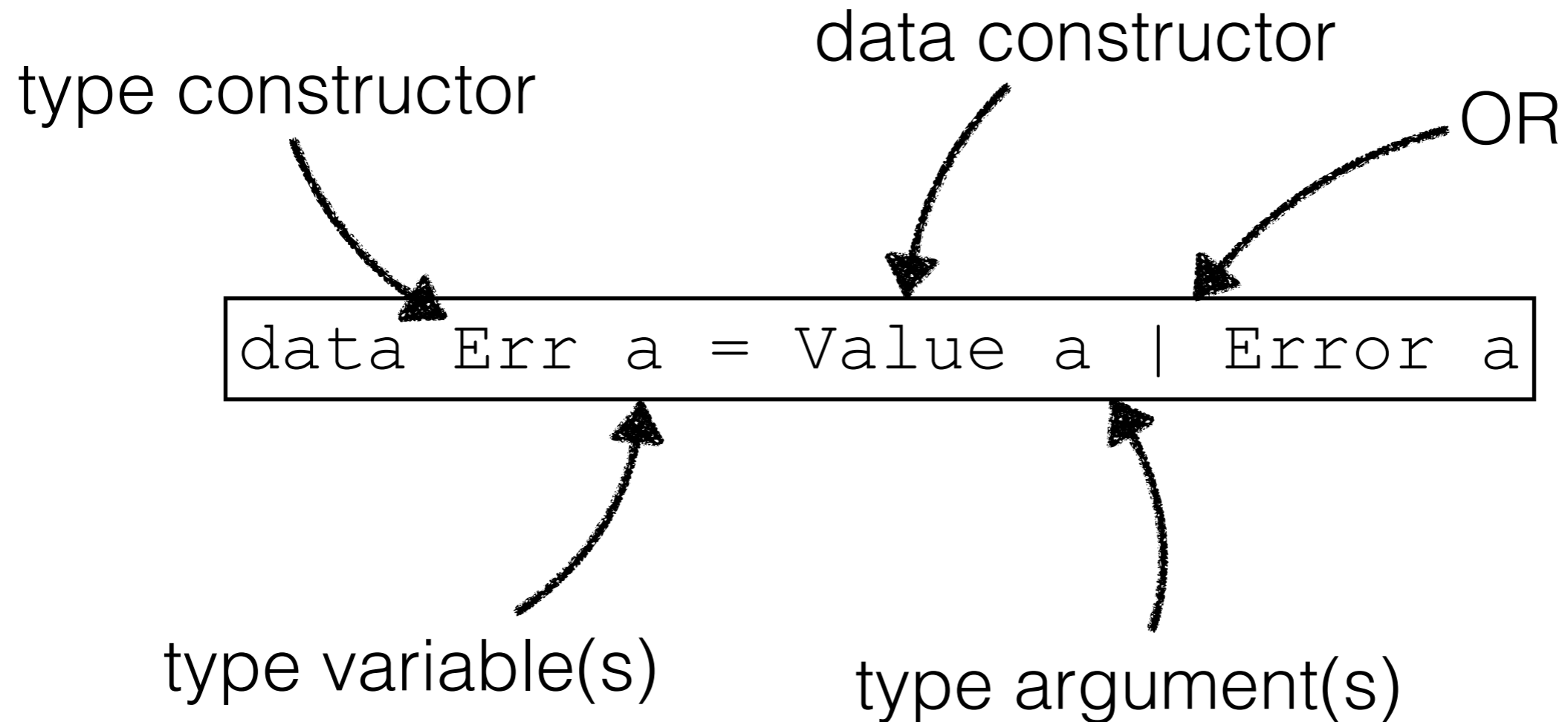
```
checkEq :: (Eq a) => a -> a -> String
checkEq x y | x == y = "Equal"
            | otherwise = "Different"
```

Why Type Classes?

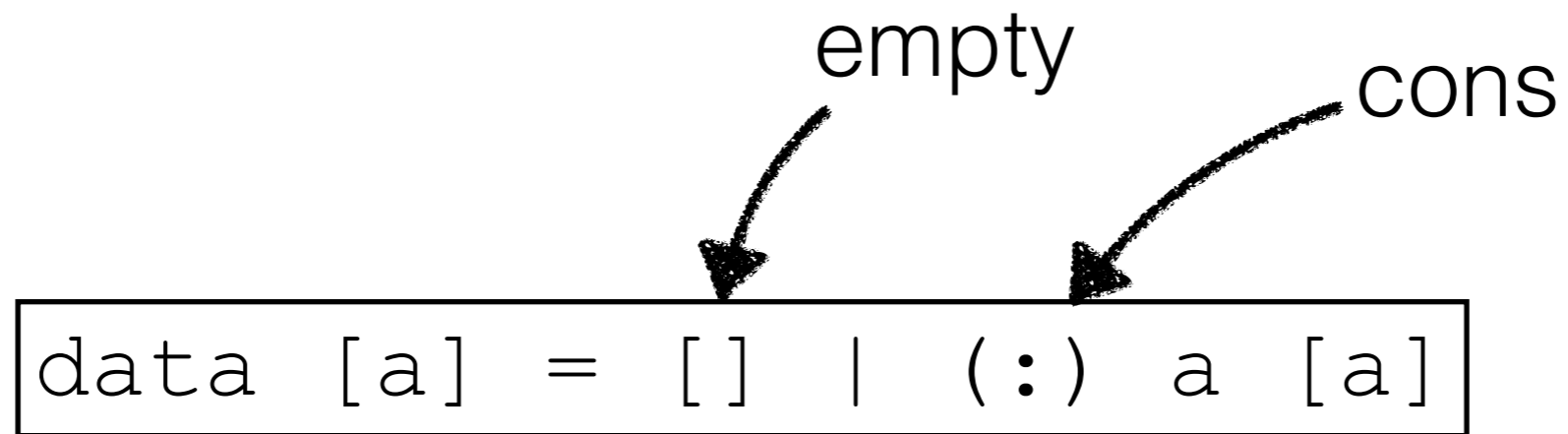
- It is an abstraction!
- Use the *same* (==) operator to things of different type
- Determine the implementation of (==) from the type
- Haskell will figure out the correct implementation
- Overloading

Data Types

Usually, a collection of labelled things



List: The Data Type



List Manipulation

Recursion

Higher Order Functions

Recursion

```
length [] = 0
length (x:xs) = 1 + length xs
```

Prove that for every list xs , $\text{length } xs \geq 0$

Induction

Prove: $\forall n \in \mathbb{N} \quad 1 + 2 + 3 + \dots + n = \frac{n \cdot (n + 1)}{2}$

If $S(0)$ and $S(n) \Rightarrow S(n+1)$, then for all n . $S(n)$

Structural Induction

<code>length []</code>	<code>= 0</code>
<code>length (x:xs)</code>	<code>= 1 + length xs</code>

Prove that for every list `xs`, `length xs >= 0`

If `S([])` and `S(xs) => S(x:xs)`, then for all `ls`. `S(ls)`

Recursion

```
length [] = 0
length (x:xs) = 1 + length xs
```

Prove that length terminates

Should functions always terminate?

No. Remember fibs?

List Manipulation

Recursion

Higher Order Functions

Higher Order Functions

```
map :: (a -> b) -> a -> b
```

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Today

Currying

List Comprehension

String Manipulation

Binary Search Trees

Monads

Type Inference: The theory break

Currying

Transform a function with many arguments
to a function with one argument

```
plus1 :: (Int, Int) -> Int  
plus1 (x, y) = x + y
```



```
plus2 :: Int -> Int -> Int  
plus2 x y = x + y
```

Currying

plus2 takes an Int and returns a *function*

```
plus2 :: Int -> Int -> Int
plus2 x y = x + y
```

If we apply the first argument we get a function

```
plus2 4 :: Int -> Int
```

Haskell Curry (1900 – 1982)



- American mathematician and logician
- Known for his work in combinatory logic
- Curry–Howard correspondence

Math

Statement

Proof

Programs

Type

Function

List Comprehension

```
filter p xs == [x | x <- xs, p x]
```

Multiplication Tables

```
[(7, 7*i) | i <- [1..10]]
```

All Combinations

```
[(i, j) | i <- [1..10], j <- [1..i]]
```


List Comprehension

Factors

Define a function `factors n` which returns a list of the integers that divide `n`. Omit the trivial factors 1 and `n`.

Examples:

```
factors 5 = []  
factors 12 = [2, 3, 4, 6]
```

```
factors n = [i | i <- [2..n-1], n `mod` i == 0]
```

List Comprehension

Pythagorean Triads

Generate a list of triples (x,y,z) such that $x^2+y^2=z^2$ and $x,y,z \leq n$.

Examples:

```
triads 5 = [(3, 4, 5), (4, 3, 5)]
```

```
triads n = [(x, y, z)
             | x<-[1..n], y<-[1..n], z<-[1..n],
             x^2+y^2==z^2]
```