Lists and Tuples Composite Data Types in Haskell

Prof Hans Georg Schaathun

Høgskolen i Ålesund

February 2, 2015

Prof Hans Georg Schaathun

Lists and Tuples

February 2, 2015 1 / 22

3 → 4 3

Image: A matrix

Tuples

# Outline









# Scalar Data Types

Int, Integer

#### 2 Double

#### Bool

Ohar, String

2

イロト イヨト イヨト イヨト

# **Composite Data**

- An adress consists of
  - Street name
  - 2 House number
  - Post code
  - Post town
- How do we represent an address?
- Tuples

# **Composite Data**

- An adress consists of
  - Street name (String)
    - House number (Integer)
  - Post code (Integer)
  - Post town (String)
- How do we represent an address?
- Tuples

3 > 4 3

# **Custom Types**

#### An address

- home = ("Larsgårdsveien",2,6025, "Ålesund")
- home :: (String, Integer, Integer, String)

#### Declare a new type alias

- type Address = (String, Integer, Integer, String)
- Type aliases is a convenience for readability

# Important

- Type name starts with a capital letter
- Function names start lower-case
- type **defines an alias**
- Type checking does not distinguish between Address and (String,Integer,Integer,String)

3 > < 3 >

### **Tuples of Tuples**

#### Tuples can be made of any types

- type Person = (String, String, Bool)
- type Address = (String, Integer, Integer, String)
- type Customer = (Person, Address)

# Programming with Tuples

#### A tuple is a single object

- getAddress :: Customer -> Address
- getAddress c = snd c
- snd returns the second element of a pair
- In fact, snd is defined as
  - snd :: (a,b) -> b
  - snd (\_,y) = y
  - This is pattern matching with tuples

### Pattern Matching

#### showCustomer :: Customer -> String

# showCustomer (p,a) = showPerson p ++ showAddress a

- showPerson :: Person -> String
- showPerson (x,y,\_) = x ++ " " ++ y ++ "\n"
- showAddress :: Address -> String
- showAddress (x,y,) = x ++ " " ++ show y ++ "\n"

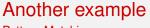
### Pattern Matching

- showCustomer :: Customer -> String
- showCustomer (p,a) = showPerson p ++ showAddress a
- showPerson :: Person -> String
- showPerson (x,y,\_) = x ++ " " ++ y ++ "\n"
- showAddress :: Address -> String
- showAddress (x,y,) = x ++ " " ++ show y ++ "\n"

### Pattern Matching

- showCustomer :: Customer -> String
- showCustomer (p,a) =
   showPerson p ++ showAddress a
- showPerson :: Person -> String
- showPerson (x,y,\_) = x ++ " " ++ y ++ "\n"
- showAddress :: Address -> String
- showAddress (x,y,) = x ++ " " ++ show y ++ "\n"

Tuples



Pattern Matching

- addPair :: (Integer, Integer) -> Integer
- addpair (x, y) = x + y

Prof Hans Georg Schaathun

Lists and Tuples

February 2, 2015 10 / 22

3

イロト イヨト イヨト イヨト

### Outline





#### Algebraic Data Types





2

イロト イヨト イヨト イヨト

# Algebraic Data Types

#### • We can declare new types type

- type Person = (String, String, Bool)
- composite types using tuples
- type aliases, not new types
- Genuinly new types is possible
  - Algebraic data types
  - data Person = Person String String Bool
  - Objects are created with the constructor Person
  - me = Person John Doe True
- We will consider algebraic data types next week

(B)

# Outline









2

# The limitation of tuples

- Each tuple type has a fixed length
  - (Integer, Integer) is a different type from (Integer, Integer, Integer)
- What if you need a list of customers, with unbounded length?
  - then lists is the answer
- [Customer] is a list type
  - arbitrary number of customers

3 > < 3 >

4 D b 4 A b

#### Lists versus tuples

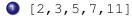
- item = ("Oranges", 5)
- goods = ["Oranges", "Bananas", "Apples"]

	Tuples	Lists
Length	Fixed length	Variable length
Constituent	Any combination	One type for all
Types		elements

Image: Image:

# Lists definitions

Some lists of type [Integer]



- 2 [1..10]
- [0, 5..100]
- [10,8..0]
- 5 []
- **6** [1..]

# Lists definitions

Some lists of type [Integer]

- 0 [2,3,5,7,11]
- 2 [1..10]
- [0, 5..100]
- [10,8..0]
- (empty)
- [1..] (infinite)

Image: A matrix and a matrix

# Functions on lists

```
let l = [2,3,5,7,11]
l!!3
head l
tail l
l ++ [13,17,19]
0:1
```

Prof Hans Georg Schaathun

э

イロト イヨト イヨト イヨト

# Functions on lists

```
let l = [2,3,5,7,11]

l!!3 \rightarrow 7

head l

tail l

l ++ [13,17,19]

0:1
```

Prof Hans Georg Schaathun

February 2, 2015 17 / 22

э

・ロト ・ 四ト ・ ヨト ・ ヨト

# Functions on lists

```
let l = [2,3,5,7,11]
l!!3 \rightarrow 7
head l \rightarrow 2
tail l
l ++ [13,17,19]
0:1
```

Prof Hans Georg Schaathun

February 2, 2015 17 / 22

э

・ロト ・ 四ト ・ ヨト ・ ヨト

#### Functions on lists

```
let l = [2,3,5,7,11]

l!!3 \rightarrow 7

head l \rightarrow 2

tail l \rightarrow [3,5,7,11]

l ++ [13,17,19]

0:1
```

Prof Hans Georg Schaathun

Lists and Tuples

February 2, 2015 17 / 22

イロト イヨト イヨト イヨト

# Functions on lists

let 1 = 
$$[2,3,5,7,11]$$
  
1!!3  $\rightarrow 7$   
head 1  $\rightarrow 2$   
tail 1  $\rightarrow [3,5,7,11]$   
1 ++  $[13,17,19] \rightarrow [2,3,5,7,11,13,17,19]$   
0:1

Prof Hans Georg Schaathun

Lists and Tuples

February 2, 2015 17 / 22

2

イロト イロト イヨト イヨト

#### Functions on lists

#### let 1 = [2, 3, 5, 7, 11]1!!3 $\rightarrow$ 7 head 1 $\rightarrow 2$ tail l $\rightarrow$ [3, 5, 7, 11] $1 ++ [13, 17, 19] \rightarrow [2, 3, 5, 7, 11, 13, 17, 19]$ 0:1 $\rightarrow$ [0,2,3,5,7,11]

Prof Hans Georg Schaathun

Lists and Tuples

17/22February 2, 2015

### The String is a List

#### ● ['a','c'..'m']

• "acegikm"

- "Hello" ++ ", " ++ "John"
   List concatenation used on strings
- 🗿 head "Hello"
- 🕘 tail "Hello"

э

・ロト ・ 四ト ・ ヨト ・ ヨト

# The String is a List

#### ● ['a','c'..'m']

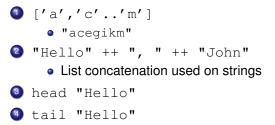
#### • "acegikm"

- "Hello" ++ ", " ++ "John"
   List concatenation used on strings
- 🗿 head "Hello"
- 🕘 tail "Hello"

э

・ロト ・ 四ト ・ ヨト ・ ヨト

#### The String is a List



Prof Hans Georg Schaathun

February 2, 2015 18 / 22

3

イロト イロト イヨト イヨト

#### • let l = [1..10]

#### Set comprehension in mathematics

• 
$$\{2x|x=1,\ldots,10\}$$

• 
$$\{2x | x \in \{1, \dots, 10\}\}$$

#### • List comprehension in Haskell

Prof Hans Georg Schaathun

э

#### Set comprehension in mathematics

• 
$$\{2x|x=1,\ldots,10\}$$

• 
$$\{2x | x \in \{1, \dots, 10\}\}$$

• List comprehension in Haskell

Prof Hans Georg Schaathun

(B)

#### Set comprehension in mathematics

• 
$$\{2x|x=1,\ldots,10\}$$

•  $\{2x | x \in \{1, \dots, 10\}\}$ 

• List comprehension in Haskell

Prof Hans Georg Schaathun

• let l = [1..10]

#### Set comprehension in mathematics

- $\{2x|x=1,\ldots,10\}$
- $\{2x | x \in \{1, \dots, 10\}\}$

#### • List comprehension in Haskell

# List comprehension with conditions

Prof Hans Georg Schaathun

Lists and Tuples

February 2, 2015 20 / 22

2

< 注 > < 注 >

Closure

### Outline









# Summary

- Three types of composite data types
  - Tuples
  - 2 Lists
  - Algebraic data types
- Function definitions with pattern matching
  - patterns give access to constituent elements

3 > < 3 >