

Intermediate Functional Programming in Haskell

Universität Bielefeld, Sommersemester 2015

Jonas Betzendahl & Stefan Dresselhaus

1 Typen

- Beispiele
- Maybe
- Either

Primitive Datentypen sind eine Annotation, wie die Bits im Speicher interpretiert werden sollen.

Primitive Datentypen sind eine Annotation, wie die Bits im Speicher interpretiert werden sollen.

Einige primitive Datentypen sollten euch aus anderen Programmiersprachen schon bekannt sein:

- Zahlen (z.B. Int, Integer, Float, Double, ...)
- Zeichenketten (z.B. String, UTF-8-Strings, ...)

Es gibt auch Datentypen höherer Ordnung. Diese zeichnen sich dadurch aus, dass sie alleine nicht vollständig sind.

Es gibt auch Datentypen höherer Ordnung. Diese zeichnen sich dadurch aus, dass sie alleine nicht vollständig sind.

Auch hier sollten schon einige bekannt sein:

(a,k,v steht hier jeweils für einen (primitiven) Datentypen)

- Liste von a
- Hashmap von k und v
- Vektor von a
- Tree von a
- Zusammengesetzte Typen (z.B. Structs in C/C++)

Es gibt auch Datentypen höherer Ordnung. Diese zeichnen sich dadurch aus, dass sie alleine nicht vollständig sind.

Auch hier sollten schon einige bekannt sein:

(a,k,v steht hier jeweils für einen (primitiven) Datentypen)

- Liste von a
- Hashmap von k und v
- Vektor von a
- Tree von a
- Zusammengesetzte Typen (z.B. Structs in C/C++)

Im folgenden gehen wir auf 2 wesentliche zusammengesetzte Typen in Haskell ein: Maybe und Either.

Einen neuen Datentypen definieren wir in Haskell mit dem Keyword `data`:

```
data Maybe a = Nothing  
             | Just a
```


Einen neuen Datentypen definieren wir in Haskell mit dem Keyword `data`:

```
data Maybe a = Nothing  
             | Just a
```

Was hat das für einen Sinn?

Einen neuen Datentypen definieren wir in Haskell mit dem Keyword `data`:

```
data Maybe a = Nothing  
             | Just a
```

Was hat das für einen Sinn?

Maybe gibt das Ergebnis einer Berechnung an, die fehlschlagen kann. In klassischen Sprachen wird hier meist ein „abgesprochener“ Fehlerzustand zurückgegeben (0, -1, null, ...). In Haskell wird dies über den Rückgabetypp deutlich gemacht.

Beispiel: Finden eines Elementes in einer Liste

Beispiel: Finden eines Elementes in einer Liste

```
ghci > import Data.List
ghci > findIndex (== 5) [1..10]
Just 4                -- [1..10] !! 4 => 5
```

```
ghci > findIndex (== 1000) [1..10]
Nothing
```

Beispiel: Finden eines Elementes in einer Liste

```
ghci > import Data.List
ghci > findIndex (== 5) [1..10]
Just 4                -- [1..10] !! 4 => 5
```

```
ghci > findIndex (== 1000) [1..10]
Nothing
```

Da wir 1000 in der Liste der Zahlen von 1-10 nicht finden können, haben wir keinen gültigen Index, daher bekommen wir ein Nothing.

foo