Haskell

safety first

Steven Kutsch

Übersicht

- rein funktional
- lazy
- sicher

Funktionen

```haskell
sortLines t = unlines (sort (lines t))

sortLines' t = (unlines . sort . lines) t

sortLines'' = unlines . sort . lines
```

mehr Funktionen

```
reverseLines = unlines . reverse . lines

first2Lines = unlines . take 2 . lines

...

byLines f = unlines . f . lines
```

Currying

```
add :: Num a => a -> a -> a
add x y = x + y

Main> add 3 4
7

Main> (add 3) 4
7
```

Currying

```
add :: Num a => a -> (a -> a)
add x y = x + y

Main> add 3 4
7

Main> (add 3) 4
7
```

PFA & Lifting

```
add3 y = 3 + y
add3' = (+) 3

add3onLists l = (map add3) l
add3onLists' = map add3
```

Daten

```
data MyList α = NIL
              | Cons α (MyList α)
                deriving Show


list1 = NIL
list2 = Cons “a” list1
list3 = Cons “b” list2
list4 = Cons “a” list4
```

Listenfunktionen

```
car :: MyList a -> a
car (Cons x xs) = x
car NIL = undefined

cdr :: MyList a -> MyList a
cdr (Cons x xs) = xs
cdr NIL = NIL
```

Listenfunktionen

```
car :: MyList a -> a
car (Cons x xs) = x
car NIL = error "Damn it!"

cdr :: MyList a -> MyList a
cdr (Cons x xs) = xs
cdr NIL = NIL
```

Maybe

```
data Maybe α = Just α
             | Nothing



car (Cons x xs) = Just x
car NIL = Nothing

cdr (Cons x xs) = Just xs
cdr NIL = Nothing
```

Maybe

```
car :: MyList a -> Maybe a
cdr :: MyList a -> Maybe (MyList a)

cadr x = (car . cdr) x
```

```
Typerror: Couldn’t match type Maybe (MyList a)
                  with expected MyList a
```

Monaden

```
(>>=) :: m a -> (a -> m b) -> m b
```

```
cadr :: MyList a -> Maybe a
cadr x = cdr x >>= car
```

mehr Haskell

- Learn you a Haskell for great good
- Real World Haskell

- tryhaskell.org

- Mehr zu Monaden
  - nondeterminism.de