KEEP CALM AND CURRY ON

Funktionales Programmieren mit Beispielen in

und

Martin Heuschober

Wien, 28. Juni 2017,

T Systems

INTRO

T Systems

ÜBERBLICK

Jun 2017	Jun 2016	Change	Programming Language	Ratings	Change
1	1		Java	14.493%	-6.30%
2	2		C	6.848%	-5.53%
3	3		C++	5.723%	-0.48%
4	4		Python	4.333%	+0.43%
5	5		C#	3.530%	-0.26%
6	9	⌃	Visual Basic .NET	3.111%	+0.76%
7	7		JavaScript	3.025%	+0.44%
8	6	⌄	PHP	2.774%	-0.45%
9	8	⌄	Perl	2.309%	-0.09%
10	12	⌃	Assembly language	2.252%	+0.13%
11	10	⌄	Ruby	2.222%	-0.11%
12	14	⌃	Swift	2.209%	+0.38%
13	13		Delphi/Object Pascal	2.158%	+0.22%
14	16	⌃	R	2.150%	+0.61%
15	48	⌃⌃	Go	2.044%	+1.83%
16	11	⌄⌄	Visual Basic	2.011%	-0.24%
17	17		MATLAB	1.996%	+0.55%
18	15	⌄	Objective-C	1.957%	+0.25%
19	22	⌃	Scratch	1.710%	+0.76%
20	18	⌄	PL/SQL	1.566%	+0.22%

Tiobe - Programmiersprachen

39	Haskell	0.425%

Tiobe - Haskell

T · · Systems

TIMELINE

History of Programming languages

SYNTAX & FEATURES

```haskell
main :: IO ()
main = putStrLn "Hello Haskell"
```

```java
public class HelloWorld {
  public static void Main (String[] args) {
    System.out.printLn("Hello Java!");
  }
}
```

- Whitespace sensitive
- Funktionsaufrufe ohne ()
- Typ-Signatur (optional) getrennt von Definition
- Funktionen sind "first level citizens"
- Starkes Typsystem + Typinferenz

- Objektorientiert + Vererbung -`NullPointer`
- Riesiges Ökosystem/Libraries Entwickler
- IDEs
- bisschen Funktional (seit Java8)

FUNKTIONALES PROGRAMMIEREN

WAS IST FUNKTIONAL?

Erst einmal ein Beispiel

```haskell
module Card where (sortBy, on, module Card)
import Data.List
import Data.Function

data Suit = Heart | Clubs | Diamond | Spades
          deriving (Eq, Show, Ord, Enum)

data Rank = Ace   | King  | Queen | Jack | Ten | Nine
          | Eight | Seven | Six   | Five | Four| Three
          | Two deriving (Eq, Show, Ord, Enum)

data Card = Card {suit :: Suit, rank :: Rank}
          deriving (Show, Eq)

allCards :: [Card]
allCards = [Card s v | s <- [Heart ..], v <- [Ace ..]]
```

```java
public enum Suit {
  Heart , Clubs , Diamond , Spades
}

public enum Rank {
  Ace , King , Queen , Jack , Ten , Nine , Eight,
  Seven , Six , Five , Four , Three , Two
}

public class Card {
  public Suit suit;
  public Rank rank;

  public class Card(Suit suit, Rank rank) {
     this.suit = suit;
     this.rank = rank;
```

T · · Systems ·

FUNKTIONAL JETZT ABER WIRKLICH

Eine Programmiersprache darf sich (meiner Meinung nach) **funktional** nennen wenn:

- Funktionen als Parameter bzw. Rückgabewerte von anderen Funktionen auftauchen können
- Funktionen in "Variablen" gesteckt werden können
- jede Funktion einen Rückgabewert hat.

Bonuspunkte gibt es für partielle Funktionsaufrufe, i.e.

- wenn eine Funktion mit mehreren Parametern mit nur einem Parameter aufrufbar ist, und eine Funktion mit um einen Parameter weniger zurückgibt.

WAS IST EINE FUNKTION

Java8

```
f :: a -> b
g :: a -> b -> c
```

T · · Systems ·

```
λ> :set -XOverloadedStrings
λ> import qualified Data.Text as T
λ> uppercased = T.toUpper "lowercase"
λ> uppercaser = T.toUpper
```

```
☕> "lowercase".toUpperCase()
☕> String::toUpperCase()
```

T-Systems

Aufgabe: (Stackoverflow) Mache aus einer Liste: `[5,4,2]` => `"+----+----+--+"`

```
String fence (List<Integer> lst){
    String result = "+";
    for( Integer i : lst) {
        for (int j = 0; j < i,j++){
          result += "-";
        }
        result += "+";
    }
    return result;
}
```

```
fence :: [Int] -> String
fence = between '+' . intercalate "+" . map (`replicate` '-')
  where between c str =  c:(str ++[c])
```

MACHE AUS EINER LISTE: [5,4,2] =>

"| + + |"

```
String fence (List<Integer> lst){
    String result = "+";
    for( Integer i : lst) {
        for (int j = 0; j < i,j++){
          result += "-";
        }
        result += "+";
    }
    return result;
}

String wall (List<Integer> lst){
    String result = "|";
    for( Integer i : lst) {
        for (int j = 0; j < i,j++){
          result += " ";
```

```
fence :: [Int] -> String
fence = between '|' . sepBy '+' . fillWith '-'

wall :: [Int] -> String
wall = between '|' . sepBy '+' . fillWith ' '

fillWith :: Char -> [Int] -> [String]
fillWith c ints = map (flip replicate c) ints

sepBy :: String -> [String] -> String
sepBy i str = intercalate i strs

between :: Char -> String -> String
between c str = c:(str++[c])
```

T Systems

```
λ> sortBy (compare `on` rank) allCards
λ> :t sortBy
sortBy :: (a -> a -> Ordering) -> [a] -> [a]
λ> :t on
on :: (b -> b -> c) -> (a -> b) -> a -> a -> c
```

```
☕> Collections.sort(allCards(),new Comparator(){..}
☕> typeOf(sortBy)
List<T> sortBy (BiFunction<T,T,int> cmp, List<T> in)
☕> typeOf(sortBy)
BiFunction<A,A,C> on (BiFunction<B,B,C> cmp, Function<A,B> f)
```

T Systems

PARTIELLE FUNKTIONSAUFRUFE

```
Employee :: Company -> Name -> Employee
tSystems = Employee "T-Systems"
map tSystems ["Martin Heuschober" ..]
map (Employee "T-Systems") ["Martin Heuschober"..]
```

```
Employee (String company, String name) {..}
Employee tSystems(String name) = Employee("T-Systems", name)
List<String> employeeNames = new ArrayList<> (["Martin Heuschobe
employeeNames.map(tSystems)
```

T · · Systems ·

HIGHER ORDER FUNCTIONS

MAP

- macht aus einer Liste eine neue Liste
- Quasi eine eingeschränkte `for`-loop†

```haskell
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map f xs
```

```java
for (Elmtclass e : lst) {
  e' = f(e)
}
```

```java
for( int i = 1, i < lst.length(), i++) {
  e' = f(lst[i]) + f(lst[i-1])
}
```

†: Ich glaube aber fast das gleiche wie eine `foreach`-loop

ABER WAS BRINGT UNS DAS?

- Parallelisation for "free"
- equational reasoning
- (stream) fusion

FOLD/REDUCE

```haskell
fold :: (a -> b -> a) -> a -> [b] -> a
fold _ acc [] = acc
fold f acc (x:xs) = fold f (f acc x) xs
```

```
for (Elmtclass e : lst) {
  acc = f(acc, e);
}
return acc;
```

T··Systems·

FILTER

```haskell
filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter p (x:xs) = let xs' = filter p xs
                  in if p x then x : xs' else xs'
```

```haskell
filter p = foldr (\y acc -> if p y then y:acc else acc) []
```

```
for (Elmtclass e : lst) {
  if (p(e)) newlist.append(e);
}
return newlist;
```

T··Systems·

ALL/ANY

```haskell
any :: (a -> Bool) -> [a] -> Bool
any _ [] = False
any p (x:xs) = if p x then True
                      else any p xs
```

```haskell
all :: (a -> Bool) -> [a] -> Bool
all _ [] = True
all p (x:xs) = p x && all p xs
```

```haskell
any p = foldr (\y acc -> p y || acc ) False
all p = foldr (\y acc -> p y && acc ) True
```

```haskell
all p = not . all (not . p)
```

```
for (Elmtclass e : lst) {
  if (p(e)) return true;
}
return false;
```

```
for (Elmtclass e : lst) {
  if (!p(e)) return false;
}
return true;
```

T-Systems

TYPSYSTEM

INFO

- Hindley-Milner
- stark, statisch

MEIN PROBLEM MIT SCHWACHEN TYPSYSTEMEN

Aus der Dokumentation von node.js (fast wortgetreu) `fs.openSync(path, flags[, mode])`

- `path <string> | <Buffer> | <URL>`
- `flags <string> | <number>`
- `mode <integer>`

```
// get the file descriptor of the file to be truncated
const fd = fs.openSync('temp.txt', 'r+');
```

Synchronous version of fs.open(). Returns an integer representing the file descriptor.

OFFENE FRAGEN

- Welche Flags gibt es?
- Was ist das '+'?
- Welche numerischen Flags gibt es?
- Sind das jetzt auch `<integer>` oder auch Kommazahlen?

- Aber ok das ist ja nur die Dokumentation - gute Programmierer kümmern sich ja gern darum, dass ihre Doku brandaktuell und vollständig ist

TYPSYSTEME SIND JA DA DEN PROGRAMMIERERN HELFEN KEINE FEHLER ZU MACHEN

fs.truncate(<integer>, <integer>, <Function>)

```
// truncate the file to first four bytes
fs.ftruncate(4, fd, (err) => {
  assert.ifError(err);
  console.log(fs.readFileSync('temp.txt', 'utf8'));
});
```

OK DAS KANN JA NICHT PASSIEREN WENN MAN DIE DOKU ORDENTLICH LIEST

machen ja auch alle - und kopieren nicht einfach Code von Stackoverflow

`fs.truncate(fd <integer>, len <integer>, callback <Function>)`

```
fs.ftruncate(fd+1,4, (err) => {
  assert.ifError(err);
  console.log(fs.readFileSync('temp.txt', 'utf8'));
});
```

T··Systems·

WOZU TYPSYSTEME ALSO?

1. Soll unterstützen.
2. Soll so viel Info wie möglich beinalten.
3. Soll so wenig einschränken wie möglich.
4. Soll weitgehend "automatisch" passieren.

Jeder Fehler der zur Compile-time erkannt wird, landet nie bei einem Kunden.

DINGE DIE DAS HASKELL TYPSYSTEM GUT MACHT

T Systems

UND ANDERE SYSTEME NICHT KÖNNEN:

IO EINGRENZEN

```
readLn :: Read a => IO a
fmap readMay . getLine :: Read a => IO (Maybe a)
withSqliteConn :: ConnString -> Query a -> IO a
```

SUM TYPES

```haskell
data Either a b = Left a | Right b

data JSON = JsonString Text
          | JsonNumber Rational
          | JsonBool   Bool
          | JsonNull
          | JsonObject (Map Text JSON)
          | JsonArray [JSON]
```

PATTERN MATCHING

```haskell
data IPv4 = IPv4 Word8 Word8 Word8 Word8
runParser :: Parser a -> String -> Either String a

case runParser ipv4 "127.-1.120.255"
  of Left msg -> do something with errormessage
     Right x -> do something with result

ipv4 :: Parser IPv4
ipv4 = do a <- check =<< decimal
          dot
          b <- check =<< decimal
          dot
          c <- check =<< decimal
          dot
          d <- check =<< decimal
          return IPv4 a b c d
  where dot :: Parser ()
        dot = void $ char '.'
        check :: Integral -> Parser Word8
        check x = do unless (0 <= x && x <= 256) $
                       fail "Failed parsing IPv4"
                     return $ fromIntegral x
```

TYPINFEREZ

```
map :: _
map f [] = []
map f (x:xx) = f x: map f xx
```

[ ] und (:) sind Listenkonstruktoren (leere Liste und `cons`-Operator) => 2tes Argument von `map` muss eine Liste sein.

```
map :: _ -> [a] -> _
```

ebenso das Ergebnis

```
map :: _ -> [a] -> [b]
```

f wird in Zeile 2 auf das erste Element der Argument-Liste angewandt => damit muss f eine Funktion sein

```
map :: (x -> y) -> [a] -> [b]
```

f wird in Zeile 2 auf das erste Element der Argument-Liste angewandt => damit muss f der Wertebereich von f a sein und der Zielbereich b

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

UND MAN SICH KLAUEN KANN

T Systems

GENERICS VERWENDEN UND ARRAYS VERMEIDEN

```
public static void main(String[] args) {
    Circle[] circles = new Circle[2];
    Shape[] arr = circles;
    arr[0] = new Circle(1.0);
    arr[1] = new Square(1.0);
    for (Shape s : arr) {
       System.out.println(s);
    }
}

public static void main(String[] args) {
    List <Circle> lst = new ArrayList<>();
    lst.add(new Circle(1.0));
    lst.add(new Square(1.0));
    for (Shape s : lst) {
       System.out.println(s.area());
    }
}
```

HASKELL IST PURE

- `NonNullable`/`@Nullable` verwenden
- IO minimieren
- mutable State minimieren - z.B. `List.append(x)` verändert eine Liste und ist daher eine schlechte Idee
    - Abstraktionen wie Funktor, Applicative, Monad, Monoid etc.

FUNDAMENTAL THEOREM OF SOFTWARE ENGINEERING

"We can solve any problem by introducing an extra level of indirection."

Butler Lampson

FUNCTOR

WHAT?

Ziel: Einen Container-Datentyp elementweise verändern, aber die Struktur des Containers beibehalten.
Warum: Fast jeder Container hat diese Eigenschaft.

HASKELL

```haskell
class Functor c where
    fmap :: (a -> b) -> c a -> c b

data [a] = ..
data Maybe a = Nothing | Just a
data Tree a = Empty
            | Tree { label :: a
                   , leftBranch  :: Tree a
                   , rightBranch :: Tree a }
data RoseTree a = RoseTree { label :: a
                           , children :: [RoseTree a]
                           }
data IntMap a = ..
```

T··Systems·

```haskell
instance Functor [] where
  fmap _ [] = []
  fmap f (x:xs) = f x : fmap f xs

instance Functor Maybe where
  fmap _ Nothing = Nothing
  fmap f (Just a) = Just (f a)

instance Functor Tree where
  fmap _ Empty = Empty
  fmap f (Tree x lB rB) = Tree (f x) (fmap f lB) (fmap f rB)

instance Functor Tree where
  fmap f (RoseTree x xs) = RoseTree (f x) (fmap (fmap f) xs)
```

JAVA

```java
import java.util.function.Function;

interface Functor<T> {
    <R> Functor<R> fmap(Function<T, R> f);
}

interface Functor<T,F extends Functor<?,?>> {
    <R> F fmap(Function<T,R> f);
}

class Identity<T> implements Functor<T,Identity<?>> {
    private final T value;

    Identity(T value) { this.value = value; }

    public <R> Identity<R> fmap(Function<T,R> f) {
```

LINKS

T Systems

HASKELL

- haskell.org
- Learn you a haskell for great good
- Real World Haskell
- WebFramework: Yesod
- Parallel and concurrent programming in haskell
- hoogle

JAVA

- Java8 - Berlin clock kata