

# Tutorium 02: Mehr Haskell

---

Paul Brinkmeier

17. November 2020

Tutorium Programmierparadigmen am KIT

- -
- Richtig, kleine Fehler
- Aufgabe nicht verstanden
- Grundansatz falsch
- Richtig!
- Richtiger Ansatz, aber unvollständig

# Heutiges Programm

---

- Übungsblatt 1/Aufgabe 1
- Wiederholung der Vorlesung
- Aufgaben zu Haskell

# Übungsblatt 1

---

## 1.1, 1.2 — pow1 und pow2

```
module Arithmetik1 where
```

```
pow1 b e | e < 0      = error "e negativ"  
         | e == 0     = 1  
         | otherwise = b * pow1 b (e-1)
```

```
pow2 b e  
  | e < 0          = error "e negativ"  
  | e == 0         = 1  
  | e `mod` 2 == 0 = pow2 (b * b) (e `div` 2)  
  | otherwise      = b * pow2 (b * b) (e `div` 2)
```

## 1.3 — pow3

```
module Arithmetik2 where

pow3 b e
  | e < 0 = error "e negativ"
  | otherwise = pow3Acc 1 b e
where
  pow3Acc acc b e
    | e == 0 = acc
    | e `mod` 2 == 0 =
      pow3Acc acc (b * b) (e `div` 2)
    | e `mod` 2 == 1 =
      pow3Acc (b * acc) (b * b) (e `div` 2)
```

## 1.4 — root

```
module Arithmetik3 where

import Arithmetik2 (pow3)

root exp r
  | exp <= 0  = error "Exponent negativ"
  | r < 0     = error "Wurzel komplex"
  | otherwise = searchRoot 0 (r + 1)
where
  searchRoot lower upper
    | upper - lower == 1 = lower
    | r < avg 'pow3' exp = searchRoot lower avg
    | otherwise          = searchRoot avg upper
  where avg = (lower + upper) 'div' 2
```



## 1.5 — isPrime

```
module Arithmetik4 where

import Arithmetik3 (root)

isPrime n = not (any (divides n) [2..root 2 n])
  where
    divides p q = p `mod` q == 0
```

## Aufgaben 2 und 3

Teilaufgaben 2 und 3 sind noch nicht korrigiert.

↪ nächste Woche

## **Wiederholung: Eingebaute Funktionen**

---

# Eingebaute Funktionen: Funktionen höherer Ordnung

- Für `[ a ]`:
  - `map :: (a -> b) -> [ a ] -> [ b ]`
  - `filter :: (a -> Bool) -> [ a ] -> [ a ]`
  - `all :: (a -> Bool) -> [ a ] -> Bool`
  - `any :: (a -> Bool) -> [ a ] -> Bool`
  - `foldl :: (b -> a -> b) -> b -> [ a ] -> b`
- Für Funktionen:
  - `(.) :: (a -> b) -> (b -> c) -> (a -> c)`
  - `($) :: (a -> b) -> a -> b`
  - `flip :: (a -> b -> c) -> (b -> a -> c)`

Schreibt ein Modul Tut02 mit:

- `import Prelude ()` — Verhindert Laden der Standardbibliothek
- `map`
- `filter`
- `squares 1` — Liste der Quadrate der Elemente von 1
- `odd`, `even` — Prüft ob eine Zahl (un-)gerade ist
- `odds`, `evens` — Liste aller (un-)geraden Zahlen  $\geq 0$
- `foldl`
- `scanl f 1` — Wie `foldl`, gibt aber eine Liste aller Akkumulatorwerte zurück
  - Bspw. `scanl (*) 1 [1, 3, 5] == [1, 3, 15]`

# Lazy Evaluation

---

- 
- Was heißt Lazy Evaluation?
  - Wieso tritt erst bei der zweiten Eingabe ein Fehler auf?

- 
- Was heißt Lazy Evaluation?
  - Wieso tritt erst bei der zweiten Eingabe ein Fehler auf?
  - $\rightsquigarrow$  Berechnungen finden erst statt, wenn es *absolut* nötig ist



# Lazy Evaluation

[//wiki.haskell.org/Lazy\\_evaluation](http://wiki.haskell.org/Lazy_evaluation):

*Lazy evaluation means that expressions are not evaluated when they are bound to variables, but their evaluation is deferred until their results are needed by other computations.*

- Auch: *call-by-name* im Gegensatz zu *call-by-value* in bspw. C
- Was bringt das?

# Lazy Evaluation

[//wiki.haskell.org/Lazy\\_evaluation](http://wiki.haskell.org/Lazy_evaluation):

*Lazy evaluation means that expressions are not evaluated when they are bound to variables, but their evaluation is deferred until their results are needed by other computations.*

- Auch: *call-by-name* im Gegensatz zu *call-by-value* in bspw. C
- Was bringt das?
- Ermöglicht arbeiten mit unendlichen Listen
- Berechnungen, die nicht gebraucht werden, werden nicht ausgeführt

# Hangman

---

- [//pbrinkmeier.de/Hangman.hs](http://pbrinkmeier.de/Hangman.hs)
- `showHangman` — Zeigt aktuellen Spielstand als `String`
- `updateHangman` — Bildet Usereingabe (als `String`) und alten Zustand auf neuen Zustand ab
- `initHangman` — Anfangszustand, leere Liste

- 
- `s` ist der Typ des Spielzustands
  - Anfänglicher Zustand: `[]` — leere Liste an Rateversuchen
  - Parameter 1: `showHangman`
  - Parameter 2: `updateHangman`
  - Parameter 3: `initHangman`

## Hangman — Beispiele

- `showHangman "Test"['e'] == ". e . . | e"`
- `showHangman "Test"['s', 'f'] == ". . s . | s f"`
- `updateHangman "f"['a'] == ['f', 'a']`