

# Vorlesung Programmierung und Modellierung

## 1 Erste Programme

François Bry

Sommersemester 2021



CC BY-NC-SA 3.0 DE

<https://creativecommons.org/licenses/by-nc-sa/3.0/de/>

Die Vorlesung ist eine Einführung in die funktionale Programmierung mit der Programmiersprache Haskell.

- ▶ Die funktionale Programmierung ist verbreitet und beeinflusst viele Programmiersprachen.
- ▶ Das Streben nach Parallelismus macht die funktionale Programmierung immer wichtiger.
- ▶ Haskell ist gut geeignet, die funktionale Programmierung zu lernen.

Alle Themen der Vorlesung sind für die Praxis relevant.

Deklarative Sichtweise macht es einfacher Programme schneller und ohne Fehler zu schreiben.  
Die Geschwindigkeit der Prozessoren in den letzten Jahren kann kaum gesteigert werden.  
>> mehrere Kerne

Nur durch Programmieren wird eine Programmiersprache gelernt.

Auf das eigene Laptop das folgende installieren:

- ▶ Haskell-Plattform

In der Vorlesung wird GHC (Glasgow Haskell Compiler) in der Version 9.0.1 verwendet. Die Shell-Kommandos sind:

- ▶ `ghci` – interaktiver Interpreter und Debugger
  - ▶ `ghc` – Übersetzer
  - ▶ `runghc` – Laufzeitsystem mit integriertem Übersetzer
- ▶ einen an Haskell angepassten Editor wie Atom, Emacs oder Sublime
- ▶ Eventuell einen IDE (Integrated Development Environment)

Am Anfang jeder Vorlesung- und Übungsstunde aufrufen:

- ▶ `ghci` in einem Terminal-Fenster  
oder  
`https://tryhaskell.org/` in einem ersten Browser-Fenster
- ▶ `https://lmu.onlinetd.de` in einem weiteren Browser-Fenster
- ▶ eventuell die lokal gespeicherten Folien der Vorlesungstunde

Die Vorlesungsfolien sind auf Uni2Work unter

`https://uni2work.ifi.lmu.de/course/S21/IfI/ProMo/file`  
abrufbar.

Warnung:

- ▶ Es wird keine Wiederholung geben.
- ▶ Die Themen der Vorlesung bauen aufeinander auf.

Es ist folglich notwendig, den wöchentlichen Stoff unter der laufenden Woche zu

- ▶ verstehen,
- ▶ lernen,
- ▶ üben.

Verstehen und lernen ohne ausreichend zu üben, reicht nicht aus, um die Prüfung zu bestehen.

Codieren soll Spaß machen.

# Die Kapitel der Vorlesung

1. Erste Programme
2. Ein- und Ausgaben
3. Rekursion
4. Auswertung
5. Typen
6. Typklassen
7. Funktionen höherer Ordnung
8. Grundlegende Funktionen für Binärbäume
9. Die Typ-Klassen Functor und Applicative (Functor)
10. Die Typ-Klasse Monad
11. Fehlerbehandlung
12. Module, Bibliotheken und Packages

- ▶ Miran Lipovaca: *Learn You a Haskell for Great Good*, 2011  
<http://learnyouahaskell.com/>  
Gut für den Einstieg
- ▶ Bryan O'Sullivan, Don Stewart, John Goerzen: *Real World Haskell*, O'Reilly, 2009  
<http://book.realworldhaskell.org/>  
Dick aber leicht verständlich
- ▶ stackoverflow  
<http://stackoverflow.com/>  
Gut für Hinweise und Erklärungen

Die Prüfungen werden

- ▶ ähnliche Aufgaben wie die Übungsblätter stellen,
- ▶ alle Themen der Vorlesung betreffen:
- ▶ kurz nach Ende der Vorlesung stattfinden.

Die Termine der Prüfung und der Nachholprüfung werden zu gegebener Zeit auf Webseite der Vorlesung auf Uni2Work angekündigt werden.

Die Fristen zur Anmeldung zur Prüfung und zur Nachholprüfung bitte einhalten!



## 1 Erste Programme

- ▶ Der Interpreter `ghci`
- ▶ Ausdrücke und Operatoren
- ▶ Definitionen, lokale Definitionen und Überschatten
- ▶ Funktionen
- ▶ Char, String und List Typen in Haskell sind sehr wichtig
- ▶ Comprehensions

Wie man aus bereits existierenden Listen neue Listen erstellen kann

# Der Interpreter ghci

In ghci aufrufen:

:? Hilfe

:quit

1

1

1 + 1

(+) 1 1

it it : Das letzte erfolgreich gerechnetes Ergebniss

1 + "1" error

it

1 - 2

1 + (-2)

1 + (2 \* 3)

1 + 2 \* 3

3 - 1 - 1

2^3

quot 4 2

quot 5 2

True && False

True && 0 error

True || False

2 == 2 True

2 = 2 Zuweisung

2 == 2.0 True

2.0 >= 1 True

2 <= 4 True

2 /= 1 True

# Ausdrücke und Operatoren

## Operator-Präzedenzen

$1 + 2 * 3$  steht für  $1 + (2 * 3)$ .  $(*)$  hat also eine höhere Präzedenz (oder bindet stärker) als  $(+)$ .

In ghci aufrufen:

Die Höhe der Stelligkeit eines Operators bestimmt, welches Operator erstens ausgeführt wird.

```
:info (+)
```

```
:info (*)
```

## Operator-Assoziativität

$3 - 1 - 1$  steht für  $(3 - 1) - 1$  nicht für  $3 - (1 - 1)$ .

$(-)$  ist also linksassoziativ.

$2^3^4$  steht für  $2^{(3^4)}$  nicht für  $(2^3)^4$ .

$(^)$  ist also rechtsassoziativ.

In ghci aufrufen:

```
:info (-)
```

```
:info (^)
```

Die folgende Ausdrücke von ghci auswerten lassen:

```
pi  
e
```

e kann in ghci wie folgt definiert werden:

```
let e = exp 1
```

# Definitionen, lokale Definitionen und Überschatten

Lokale Definition

```
let e = let one = 1 in exp one
```

Überschatten

```
let e = let e = 1 in exp e
```

## Quiz Lokale Definition und Überschatten

`let e = 1 in exp e`

- A. Der Ausdruck hat einen Wert. ✓
- B. Der Ausdruck hat keinen Wert.

Antworten: <https://lmu.onlinetted.de>

## Quiz Stelligkeit

```
Prelude> let plusEins x = x + 1  
Prelude> let quadrat x = x^2  
Prelude> quadrat plusEins 0
```

- A. Der letzte Ausdruck hat einen Wert.
- B. Der letzte Ausdruck hat keinen Wert. ✓

Antworten: <https://lmu.onlinetted.de>

# Funktionen

Anfang eines Blocks



```
Prelude>:{  
Prelude|let f x = let quadrat y = y^2 in  
                (quadrat x) + 3  
Prelude|let g x = let plusEins y = y + 1  
Prelude|                quadrat y = y^2  
Prelude|                in quadrat (plusEins x)  
Prelude|:}
```

Spaces are very Important

//

Einrückung



# Char, String und List

Die folgenden ghci-Kommandos aufrufen:

```
:type 'a'      Char
:type "abc"    [Char]
:type 'abc'    error
:type "a"      [Char]
:type ['a', 'b', 'c'] [Char]
```

Die folgenden Ausdrücke von ghci auswerten lassen:

"ab"	last "abc"
['a', 'b']	last [1,2,3]
"ab" == ['a', 'b'] True	head "abc"
[1,2,3] ++ [4,5]	head [1,2,3]
"abc" ++ "de"	tail "abc"
reverse [1,2,3]	tail [1,2,3]
reverse "abc"	length "abc"
	length [1,2,3]

## Quiz String I


In Haskell ist jedes String eine Liste.

- A. korrekt
- B. falsch  'a' is a String but not a list

Antworten: <https://lmu.onlinetted.de>

## Quiz String II

In Haskell ist jede Liste ein String.

- A. A. korrekt  the type of a list is always [Char]
- B. B. falsch.

Antworten: <https://lmu.onlinetted.de>

# Comprehensions

Die folgenden Ausdrücke von ghci auswerten lassen:

```
let natZahlen = [0..]  
let ungeradeZahlen = [x | x <- natZahlen, x 'mod' 2 /= 0]  
take 1000 ungeradeZahlen
```

Im Ausdruck `[x | x <- natZahlen, x 'mod' 2 /= 0]` stehen

▶ `<-` für  $\in$


▶ `/=` für  $\neq$

Die verzögerte Auswertung (lazy evaluation) ermöglicht, mit unendlichen Listen wie `[0..]` zu programmieren.

## Quiz Comprehension

Was liefert der Interpreter ghci, wenn er das folgende auswertet?

```
let f n = n `mod` 2 /= 0 &&  
      n `mod` 3 /= 0 &&  
      n `mod` 5 /= 0  
let l = [n | n <- [0..10], f n]  
any odd l
```

- A. 1
- B. [1]
- C. True 
- D. False

Antworten: <https://lmu.onlinetted.de>

## 1 Erste Programme

- ▶ Der Interpreter ghci
- ▶ Ausdrücke und Operatoren
- ▶ Definitionen, lokale Definitionen und Überschatten
- ▶ Funktionen
- ▶ Char, String und List
- ▶ Comprehensions