

Einführung in Haskell und seine Werkzeuge

PD Dr. David Sabel

Goethe-Universität Frankfurt am Main

29. September 2015

Funktionale Programmiersprachen

- Programm = Menge von Funktionsdefinitionen
- Ausführung = Auswerten eines Ausdrucks
- Resultat = ein einziger Wert
- **keine Seiteneffekte** (sichtbare Speicheränderungen)!
(insbes.: keine Programmvariablen, Zuweisung, ...)



Haskell

- die pure funktionale Programmiersprache
- relativ neu: erster Standard 1990
- Benannt nach dem amerik. Mathematiker *Haskell B. Curry* (1900 - 1982)
- Haskell 98 veröffentlicht 1999, Revision 2003
- Haskell 2010, veröffentlicht Juli 2010

Die Informationsquelle: <http://haskell.org>

The screenshot shows the Haskell Language website homepage in a browser. The browser's address bar shows 'https://www.haskell.org'. The website has a dark blue header with navigation links: Haskell, Downloads, Community, Documentation, and News. The main content area features the Haskell logo and the tagline 'Declarative, statically typed code.' Below this, there is a code snippet for a sieve function. The page is divided into sections: 'Try it' with a text input field, 'Got 5 minutes?' with a tutorial link and a code sandbox, and a footer image with the text 'An open source community effort for over 20 years'.

Haskell Language

https://www.haskell.org

Haskell Downloads Community Documentation News

Haskell

An advanced purely-functional programming language

Declarative, statically typed code.

```
primes = sieve [2..]
  where sieve (p:xs) =
        p : sieve [x | x <- xs, x `mod` p /= 0]
```

Try it

Type Haskell expressions in here.

λ

Got 5 minutes?

Type `help` to start the tutorial.

Or try typing these out and see what happens (click to insert):

```
23 * 36 or reverse "hello" or foldr (:) [] [1,2,3] or
do line <- getLine; putStrLn line or readfile "/welcome"
```

These IO actions are supported in this sandbox.

An open source community effort for over 20 years

Haskell-Interpreter und Compiler

Standard-Compiler: The Glasgow Haskell Compiler (GHC)

- hochoptimierter Compiler für Haskell
- beinhaltet auch einen Interpreter (GHCi)



Helium (Universität Utrecht, NL):

- Abgespeckte Haskell-Variante, speziell zum Haskell-Lernen
- Vorteil: Bessere Fehlermeldungen
- Nachteil: Syntax ist nicht immer kompatibel!

Entwicklungsumgebungen

Es gibt einige:

- Leksah: <http://leksah.org>
- EclipseFP: <http://eclipsefp.github.io/>
- ...


Zum Einstieg und Ausprobieren reicht aus (sogar besser):

- Texteditor mit Syntax-Highlighting
- Interpreter GHCi

Einige freie Editoren

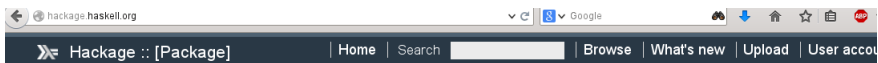
- [kate](http://kate-editor.org/) (kate-editor.org/) KDE
- [gedit](http://projects.gnome.org/gedit/) (projects.gnome.org/gedit/) Gnome
- [Notepad++](http://notepad-plus-plus.org/) (notepad-plus-plus.org/) für Windows
- [TextWrangler](http://barebones.com/products/textwrangler/) (barebones.com/products/textwrangler/) für Mac OS X
- [Emacs](http://www.gnu.org/software/emacs/) (www.gnu.org/software/emacs/)

Da Haskell Einrückungen **interpretiert**:

- 1  = 8 Leerzeichen
- **dringend empfohlen:**
Editor so **einstellen**, dass Tabulatoren automatisch durch Leerzeichen ersetzt werden.

Programmbibliotheken

Zentrale Sammelstelle: **Hackage** <http://hackage.haskell.org/>



Welcome to Hackage!

Hackage is the Haskell community's central package archive of open source software. Package authors use it to publish their libraries and programs while other Haskell programmers use tools like `cabal-install` to download and install packages (or people get the packages via their distro).

This web interface to Hackage lets you:

- **Browse** the packages (sorted by category)
- **Search** for packages by keyword (in the name or description)
- See what packages have been **uploaded recently**
- **Upload** your own packages to Hackage (note that you'll need an **account**)

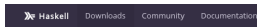
Each package includes:

- A description of what it does
- Licence information
- Author information
- A downloadable gzipped tarball
- A list of modules in the package
- Haddock documentation (if available) with source links

Paketverwaltungs-Werkzeug: **Cabal** <http://www.haskell.org/cabal/>

Haskell Platform

<http://haskell.org/platform/>



Downloads

Compiler and base libraries

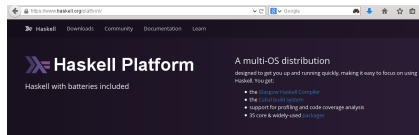
Downloads are available on a per operating system basis:

- [Windows](#)
- [OS X](#)
- [Linux](#)

Haskell Platform

Many now recommend just using a bare compiler combined with new users. However, others prefer to start with the curated bundle which is available for Windows, OS X, and Linux.

[Get the Haskell Platform →](#)



Let's get started

Bedienung des GHCi

```
> ghci ↵
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help
Prelude> 1+1 ↵
2
Prelude> 3*4 ↵
12
Prelude> 15-6*3 ↵
-3
Prelude> -3*4 ↵
-12
Prelude> 1+2+3+4+ ↵
<interactive>:1:8: parse error (possibly incorrect indentation)
Prelude> :quit ↵
Leaving GHCi.
>
```





Einige Interpreterkommandos




- `:quit` Verlassen des Interpreters
- `:help` Übersicht über die verfügbaren Kommandos
- `:load Dateiname` Lädt Haskell-Quellcode der entsprechenden Datei, die Dateiendung von *Dateiname* muss `.hs` lauten.
- `:reload` Lädt die aktuelle geladene Datei erneut

Kniffe im GHCi

- Mit `it` (für "es") erhält man das letzte Ergebnis:

```

Prelude> 1+1 
2
Prelude> it 
2
Prelude> it+it 
4
Prelude> it+it 
8
  
```

- History des GHCi mit Pfeiltasten  und 
- Auto-Completion mit 

Haskell-Quellcode

Textdatei

- im Editor erstellen
- Endung: `.hs`
- Editor so einstellen, dass **Tabulatoren durch Leerzeichen** ersetzt werden
- Auf die **Dateiendung** achten

Quellcode laden


Datei hallowelt.hs

```
1 wert = "Hallo Welt!"
```

hallowelt.hs

```
> ghci   
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help  
Prelude> :load hallowelt.hs   
[1 of 1] Compiling Main  ( hallowelt.hs, interpreted )  
Ok, modules loaded: Main.  
*Main>
```

alternativ

```
> ghci hallowelt.hs   
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help  
[1 of 1] Compiling Main  ( hallowelt.hs, interpreted )  
Ok, modules loaded: Main.  
*Main>
```

Funktionsnamen

müssen mit einem Kleinbuchstaben oder dem Unterstrich `_` beginnen, sonst

```
1  
2  
3 Zwei_mal_Zwei = 2 * 2
```

grossKleinschreibungFalsch.hs

```
Prelude> :load grossKleinschreibungFalsch.hs  
[1 of 1] Compiling Main ( grossKleinschreibungFalsch.hs )  
  
grossKleinschreibungFalsch.hs:3:1:  
    Not in scope: data constructor 'Zwei_mal_Zwei'  
Failed, modules loaded: none.
```

Kommentare

Wie kennzeichnet man etwas als Kommentar in Haskell?

- Zeilenkommentare: `--` Kommentar...
- Kommentarblöcke: Durch `{-` Kommentar `-}`

```
wert = "Hallo Welt" -- ab hier ist ein Kommentar bis zum Zeileende
wert2 = "Nochmal Hallo Welt"
-- Diese ganze Zeile ist auch ein Kommentar!
```

```
{- Hier steht noch gar keine Funktion,
   da auch die naechste Zeile noch im
   Kommentar ist
wert = "Hallo Welt"
   gleich endet der Kommentar -}

wert2 = "Hallo Welt"
```


Typen

In Haskell hat jeder Ausdruck (und Unterausdruck) einen

Typ


- Typ = Art des Ausdrucks
z.B. Buchstabe, Zahl, Liste von Zahlen, Funktion, ...
- Die Typen müssen **zueinander passen**:
Z.B. **verboten**

```
1 + "Hallo"
```

Die Typen passen nicht zusammen (Zahl und Zeichenkette)

Typen (2)

Im GHCi Typen anzeigen lassen:





```
Prelude> :type 'C'   
'C' :: Char
```

Sprechweisen:

- „'C' hat den Typ Char“
- „'C' ist vom Typ Char“
- „'C' gehört zum Typ Char“
- Char ist der Typ in Haskell für Zeichen (engl. Character)
- Typnamen beginnen immer mit einem Großbuchstaben

Typen (3)

- Im GHCi: `:set +t`
führt dazu, dass mit jedem Ergebnis auch dessen Typ gedruckt wird.

```
Prelude> :set +t   
Prelude> 10+6   
16  
it :: Num a => a  
Prelude> length "Hallo"   
5  
it :: Int  
Prelude> 5 > 3   
True  
it :: Bool
```

Typen (4)

- Der Typ `Integer` stellt beliebig große ganze Zahlen dar
- Man kann Typen auch selbst angeben:

Schreibweise *Ausdruck* :: *Typ*

```
*Main> 'C' :: Char
```

```
'C'
```

```
it :: Char
```

```
*Main> 'C' :: Integer
```

```
<interactive>:1:0:
```

```
Couldn't match expected type 'Integer'
against inferred type 'Char'
```

```
In the expression: 'C' :: Integer
```

```
In the definition of 'it': it = 'C' :: Integer
```