Einführung in Haskell und seine Werkzeuge

PD Dr. David Sabel

Goethe-Universität Frankfurt am Main

29. September 2015

Funktionale Programmiersprachen

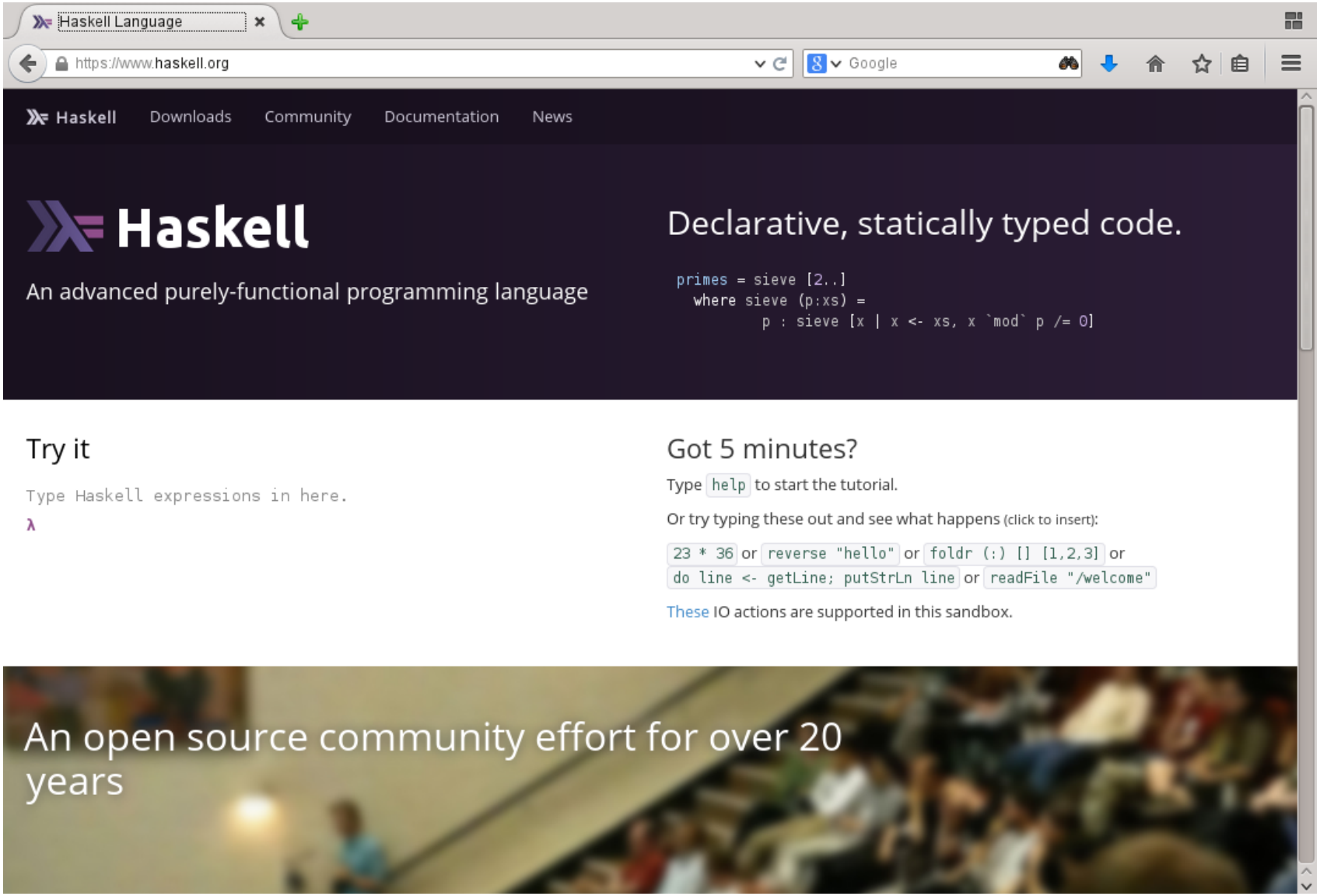
- Programm = Menge von Funktionsdefinitionen
- Ausführung = Auswerten eines Ausdrucks
- Resultat = ein einziger Wert
- keine Seiteneffekte (sichtbare Speicheränderungen)!
  (insbes.: keine Programmvariablen, Zuweisung, . . .)

Haskell

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**


Haskell Language
https://www.haskell.org
Google
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

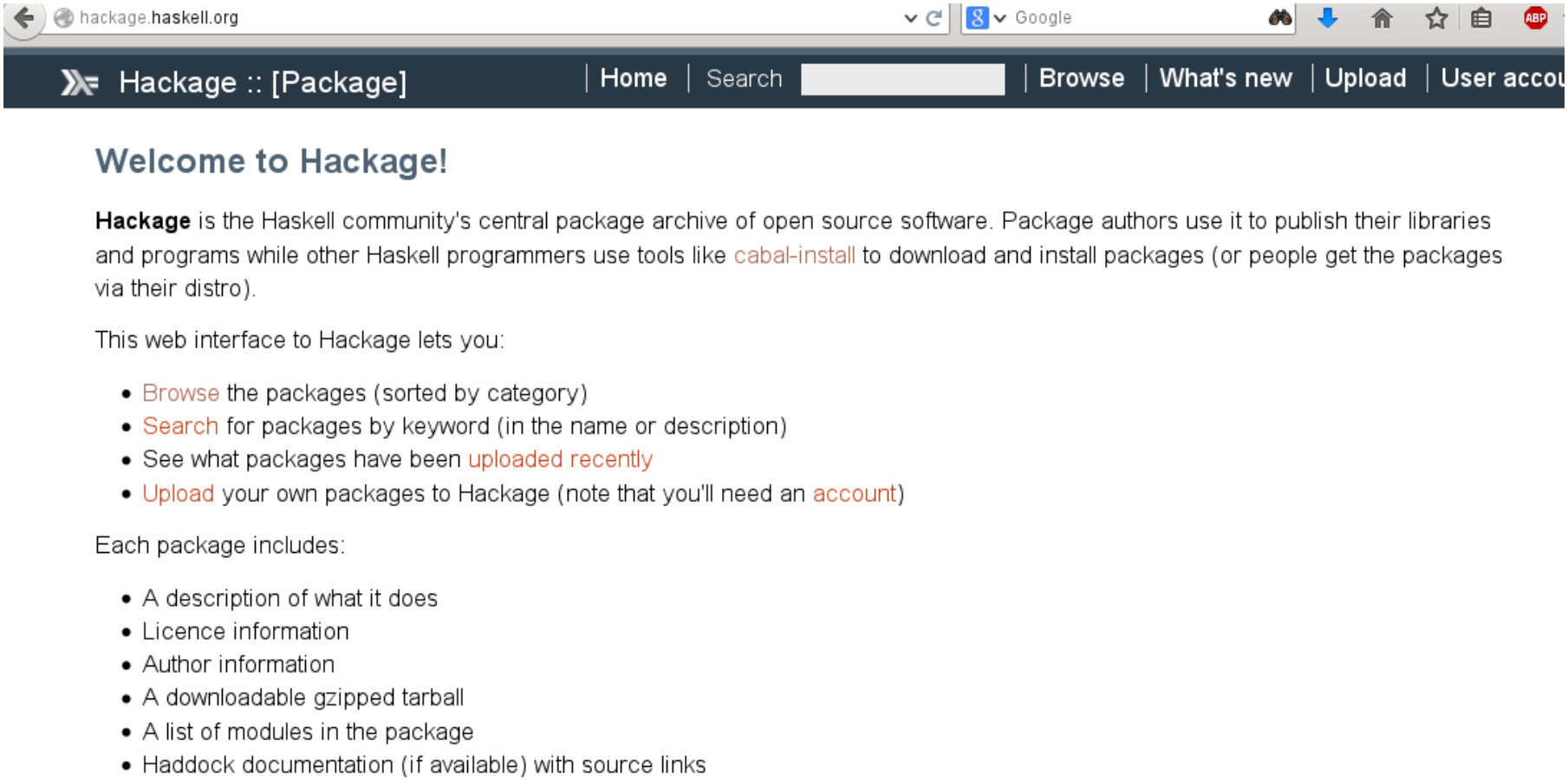
- `kate` (`kate-editor.org/`) KDE
- `gedit` (`projects.gnome.org/gedit/`) Gnome
- Notepad++ (`notepad-plus-plus.org/`) für Windows
- TextWrangler (`barebones.com/products/textwrangler/`) für Mac OS X
- Emacs (`www.gnu.org/software/emacs/`)

Da Haskell Einrückungen **interpretiert**:

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

Programmbibliotheken

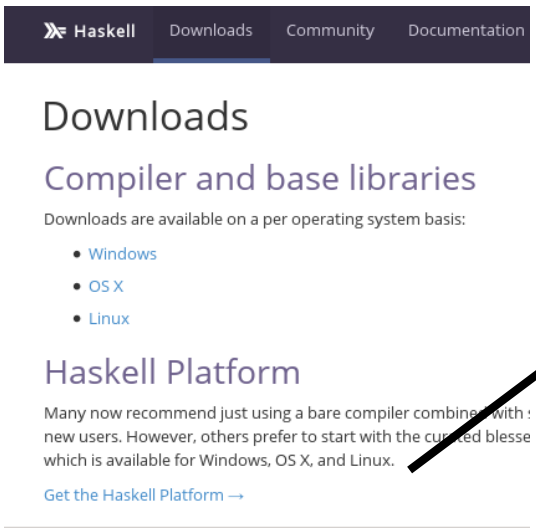
Zentrale Sammelstelle: **Hackage** `http://hackage.haskell.org/`

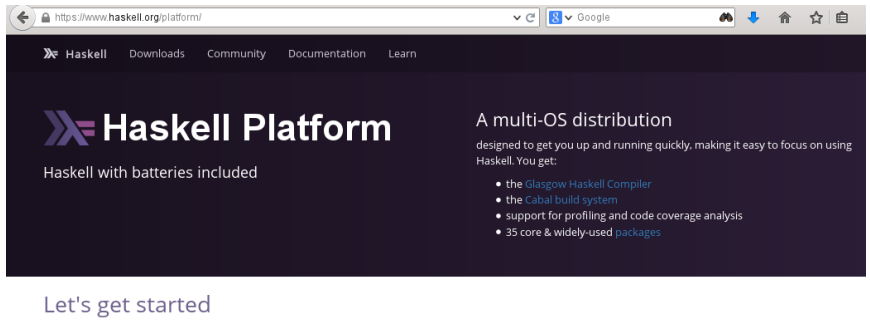


Paketverwaltungs-Werkzeug: **Cabal** `http://www.haskell.org/cabal/`

Haskell Platform

`http://haskell.org/platform/`





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
```