Hugs: Der Haskell-Interpretierer...

Aufruf von Hugs: **hugs <fileName>**

...und im Fall von FirstScript für <fileName> z.B. weiter mit

```
Main> double (sum17and4)
42
```

Wichtige Kommandos in Hugs:

:?	Liefert Liste der Hugs-Kommandos
:load <fileName>	Lädt die Haskell-Datei <fileName> (erkennbar an Endung .hs bzw. .lhs)
:reload	wiederholt letztes Ladekommando
:quit	Beendet den aktuellen Hugs-Lauf
:info name	Liefert Information über das mit name bezeichnete "Objekt"
:type exp	Liefert den Typ des Argumentausdrucks exp
:edit <fileName>.hs	Öffnet die Datei <fileName>.hs enthaltende Datei im voreingestellten Editor
:find name	Öffnet die die Deklaration von name im voreingestellten Editor
!<com>	Ausführen des Unix- oder DOS-Kommandos <com>

...mehr dazu: http://www.haskell.org/hugs/

Fehlermeldungen&Warnungen in Hugs

- Fehlermeldungen
  - Syntaxfehler
    `Main> sum17and4 == 21)` ...liefert
    `ERROR: Syntax error in input (unexpected ‘)’)`
  - Typfehler
    `Main> sum17and4 + False` ...liefert
    `ERROR: Bool is not an instance of class ‘‘Num’’`
  - Programmfehler
    ...später
  - Modulfehler
    ...später
- Warnungen
  - Systemmeldungen
    ...später

(Ausgew.) Elementare Datentypen (1)

Wahrheitswerte

Typ	`Bool`	Wahrheitswerte
Konstanten	`True :: Bool`	Symbol für "wahr"
	`False :: Bool`	Symbol für "falsch"
Operatoren	`&& :: Bool -> Bool -> Bool`	logisches und
	`|| :: Bool -> Bool -> Bool`	logisches oder
	`not :: Bool -> Bool`	logische Negation

Elementare Datentypen (2)

Ganze Zahlen

Typ	`Int`	Ganze Zahlen(endl. Ausschn.)
Konstanten	`0 :: Int`	Symbol für “0”
	`-42 :: Int`	Symbol für “-42”
	`2147483647 :: Int`	Symbol für “maxInt”
	...	
Operatoren	`+ :: Int -> Int -> Int`	Addition
	`* :: Int -> Int -> Int`	Multiplikation
	`^ :: Int -> Int -> Int`	Exponentiation
	`- :: Int -> Int -> Int`	Subtraktion (Infix)
	`- :: Int -> Int`	Vorzeichenwechsel (Prefix)
	`div :: Int -> Int -> Int`	Division
	`mod :: Int -> Int -> Int`	Divisionsrest
	`abs :: Int -> Int`	Absolutbetrag
	`negate :: Int -> Int`	Vorzeichenwechsel

Elementare Datentypen (3)

Ganze Zahlen (fortgesetzt)

Relatoren	`> :: Int -> Int -> Bool`	echt größer
	`>= :: Int -> Int -> Bool`	größer gleich
	`== :: Int -> Int -> Bool`	gleich
	`/= :: Int -> Int -> Bool`	ungleich
	`<= :: Int -> Int -> Bool`	keiner gleich
	`< :: Int -> Int -> Bool`	echt kleiner

...die Relatoren == und /= sind auf Argumente beliebiger Typen anwendbar, beispielsweise auch auf Wahrheitswerte (Stichwort: Overloading!)

Elementare Datentypen (4)

Zeichen

Typ	`Char`	Zeichen (Literal)
Konstanten	`'a' :: Char`	Symbol für “a”
	`...`	
	`'Z' :: Char`	Symbol für “Z”
	`'\t' :: Char`	Tabulator
	`'\n' :: Char`	Neue Zeile
	`'\\' :: Char`	Symbol für “backslash”
	`'\'' :: Char`	Hochkomma
	`'\"' :: Char`	Anführungszeichen
Operatoren	`ord :: Char -> Int`	Konversionsfunktion
	`chr :: Int -> Char`	Konversionsfunktion

Elementare Datentypen (5)

Gleitkommazahlen

Typ	`Float`	Gleitkommazahlen (endlicher Ausschnitt)
Konstanten	`0.123 :: Float`	Symbol für “0,123”
	`-42.4711 :: Float`	Symbol für “-42,4711”
	`123.6e-2 :: Float`	$123{,}6 \times 10^{-2}$
	...	
Operatoren	`+ :: Float -> Float -> Float`	Addition
	`* :: Float -> Float -> Float`	Multiplikation
	...	
	`sqrt :: Float -> Float`	(pos.) Quadratwurzel
	`sin :: Float -> Float`	sinus
	...	
Relatoren	`== :: Float -> Float -> Bool`	gleich
	`/= :: Float -> Float -> Bool`	ungleich
	...	

Elementare Datentypen und mehr...

Zeichenreihen

Typ	`String`	Zeichenreihen
	`type String = [Char]`	Deklaration (als Liste von Zeichen)
Konstanten	`"Haskell" :: String`	Zeichenr. für “Haskell”
	`""`	Leere Zeichenreihe
	...	
Operatoren	`++ :: String -> String -> String`	Konkatenation
Relatoren	`== :: String -> String -> Bool`	gleich
	`/= :: String -> String -> Bool`	ungleich

Ausdrücke mit Typen

5	::	Int	-- ganze Zahl (32-bit)
	::		(Haskell-Report: $-2^{29}..2^{29}-1$)
5	::	Integer	-- ganze Zahl (beliebig)
'a'	::	Char	-- Zeichen
[1,2,3]	::	[Integer]	-- Liste ganzer Zahlen
('b',4)	::	(Char, Integer)	-- Paar bzw. Tupel
\x->x+1	::	Integer -> Integer	-- Funktion
\x y->(x,y)	::	a -> b -> (a,b)	-- generische Funktion
\x->x+1	::	Num a=>a->a	-- Typ-/Klasseneinschränkung

Funktionsdefinition mit Typdeklaration

...in `.hs` Datei:

```
e :: Float
e = 2.71828

simple         ::  Integer->Integer->Integer->Integer
simple x y z   =   x * (y + z)

fact     ::  Integer -> Integer
fact n   =   if n == 1 then 1 else n * fact (n - 1)
```

Auf Einrückungen achten! Stichwort: Abseitsregel!

Auswerten von Ausdrücken und Funktionen...

Fundamental...

- Expandieren
- Simplifizieren

Einfache Ausdrücke und Funktionen (1)

$3 * (9 + 5) \Rightarrow 3 * 14 \Rightarrow 42$ oder

$3 * (9 + 5) \Rightarrow 3 * 9 + 3 * 5 \Rightarrow 27 + 3 * 5 \Rightarrow 27 + 15 \Rightarrow 42$

simple x y z = x * (y + z)

simple 3 9 5 $\Rightarrow 3 * (9 + 5) \Rightarrow 3 * 14 \Rightarrow 42$

Einfache Ausdrücke und Funktionen (2)

```
fact n = if n == 1 then 1 else (n * fact (n - 1))
```

$$
\begin{aligned}
\text{fact } 3 &\Rightarrow \text{if } 3 == 1 \text{ then } 1 \text{ else } (3 * \text{fact } (3 - 1)) \\
&\Rightarrow 3 * \text{fact } (3 - 1) \\
&\Rightarrow 3 * \text{fact } 2 \\
&\Rightarrow 3 * (\text{if } 2 == 1 \text{ then } 1 \text{ else } (2 * \text{fact } (2 - 1))) \\
&\Rightarrow 3 * (2 * \text{fact } (2 - 1)) \\
&\Rightarrow 3 * (2 * \text{fact } 1) \\
&\Rightarrow 3 * (2 * (\text{if } 1 == 1 \text{ then } 1 \text{ else } (1 * \text{fact } (1 - 1)))) \\
&\Rightarrow 3 * (2 * 1) \\
&\Rightarrow 3 * 2 \\
&\Rightarrow 6
\end{aligned}
$$

Zur Syntax von Haskell... hier Namen

Name ist — Buchstabe gefolgt von Buchstaben und Ziffern

— Infix-Operator als Folge von Sonderzeichen

Funktionsnamen, Variablen beginnen mit Kleinbuchstaben — Großbuchstaben für Typen, Konstruktoren, Module reserviert

Leerzeichen und Klammern zur Trennung von Namen

Funktionsargumente durch Leerzeichen getrennt

Leerzeichen binden stärker als Operatoren

Trennung: — Funktionsdef., Typdeklarationen in Dateien

— Auswertung von Ausdrücken im Interpreter

Currying

Obwohl es syntaktisch nur einstellige Funktionen gibt, lassen sich damit auch mehrstellige Funktionen realisieren:

$$
\begin{aligned}
\backslash x\ y\ z{-}{>}(x * (y + z))\ 3\ 9\ 5 &\Rightarrow \backslash y\ z{-}{>}(3 * (y + z))\ 9\ 5 \\
&\Rightarrow \backslash z{-}{>}(3 * (9 + z))\ 5 \\
&\Rightarrow 3 * (9 + 5) \\
&\Rightarrow 3 * 14 \\
&\Rightarrow 42
\end{aligned}
$$

Schrittweise Berechnung direkt darstellbar:

$$
\begin{aligned}
\text{simple} &= \backslash x\ y\ z{-}{>}(x * (y + z)) \\
\text{simple}\ x &= \backslash y\ z{-}{>}(x * (y + z)) \\
\text{simple}\ x\ y &= \backslash z{-}{>}(x * (y + z)) \\
\text{simple}\ x\ y\ z &= x * (y + z)
\end{aligned}
$$