1 Haskell

1.1 Haskell Typen

Float	Gleitkomma-Zahlen mit einfacher Genauigkeit 32bit
Double	Gleitkomma-Zahlen mit doppelter Genauigkeit 64bit
Int	beschränktes Ganzzahl
Integer	unbeschränkte Ganzzahl
Bool	Wahrheitswerte true/false
Char	Zeichen
Type	Liste von Typen
(TypeA, TypeB)	Paar von Typen Tupel
TypeA -> TypeB	Typenfunktion

1.2 Haskell Funktion

```
f :: X -> Y
```

1.2.1 Beispiel

```
sqrt :: Float -> Float
first :: (String, Int) -> String
second :: (String, Int) -> Int
not :: Bool -> Bool
and :: [Bool] -> Bool
logBase :: Float -> Float -> Float
```

1.2.2 Anwendung

So ruft man die Funktion auf:

```
f x
```

1.2.3 Funktionsaufbau sich ansehen

```
:t Funktionsname
```

1.3 Operatoren

```
(op) :: X -> Y -> Z
```

1.3.1 Schreibweise

```
(op) x y
x op y
```

1.4 Funktion definieren

```
f :: Int -> Int
f x = x * x + x
```

ist dasselbe wie:

$$f : \mathbb{Z} \to \mathbb{Z}$$
$$f(x) = x^2 + x$$

1.5 Datei Laden

```
:l filename
```

2 Haskell Pattern Matching

2.1 Verschiedene Pattern

- jedes einzelnte Pattern ist einzelnt
- es kann mehrere Patterns angegeben werden
- von oben nach unten

2.1.1 Beispiel

```
-- negation
neg :: Bool -> Bool
neg False = True

-- Konjunktion
(<&>) :: Bool -> Bool -> Bool
(<&>) True True = True
(<&>) _ _ = False
```

2.2 Unterstriche

Kann genutzt werdeb un irgendeine Random variable anzugeben.

3 Haskell Alternativen

3.1 If - Else

$$\text{abs}(x) = \begin{cases} -x & x < 0 \\ x & \text{sonst} \end{cases}$$

3.1.1 Erste Art das umzusetzen

```
absolute :: Int -> Int
absolute x = if x < 0 then -x else x
```

3.1.2 Mehrere Conditions

```
absolute :: Int -> Int
absolute x
    | x < 0 = -x
    | otherwise = x
```

4 Haskell Rekursion

4.1 Beispiel

$$\text{heron}_n : \mathbb{R}^+ \to \mathbb{R}^+$$

$$\text{heron}(n, a) = \begin{cases} (\text{heron}(n-1, a) + a/\text{heron}(n-1, a))/2 & n > 0 \\ a & \text{sonst} \end{cases}$$

```
heronA :: (Int, Double) -> Double
heronA n a
    | n > 0 = (heronA(n - 1, a) + a / heronA(n - 1, a))/2
    | otherwise = a
```

4.2 where - Operator

Haskell bietet nicht die Möglichkeit Speicherplatz zu reservieren
Aber man kann where verwenden für lokale Platzhalter definitionen

```
heronA :: (Int, Double) -> Double
heronA n a
    | n > 0 = (x + a / x)/2
    | otherwise = a
    where x = heronA(n - 1, a)
```

4.3 error-Handling

```
fibC :: Int -> Int
fibC 0 = 0
fibC 1 = 0
fibC n
    | n < 0 = error "illegal argument"
    | otherwise = fibC(n - 1) + fibC(n - 2)
```

4.4 Um -7 in der Konsole zu benutzen beispielsweise

```
fibC $ -7
```