Crashkurs: Haskell

Mentoring FU Berlin 7.11.2018

Felix Droop

Crashkurs Haskell

Freie Universität Berlin

1. GHCi Umgebung <-
2. Konstanten und Datentypen
3. Funktionen
4. Fallunterscheidung
5. Rekursion

Glasgow Haskell Compiler interactive

- www.haskell.org/downloads
- Haskell Platform installieren
- Texteditor öffnen
- dateiname.hs Dokument erstellen

```
-- Haskell Crashkurs
-- 07.11.2018
-- Autor: Felix Droop
```

} Kommentare

- `ghci> :load dateiname.hs / :l dateiname.hs`

Crashkurs Haskell

1. GHCi Umgebung
2. Konstanten und Datentypen <-
3. Funktionen
4. Fallunterscheidung
5. Rekursion

Die Konstante


```haskell
-- Haskell Crashkurs
-- 07.11.2018
-- Autor: Felix Droop

a :: Int
a = 42
a = 43
```

<- Signatur

<- Wert

- Haskell ist eine deklarative Programmiersprache

Primitive Datentypen

Freie Universität Berlin

```haskell
yes_or_no :: Bool
yes_or_no = True

zahl :: Int
zahl = 42

grosse_zahl :: Integer
grosse_zahl = 420000

kommazahl :: Float
kommazahl = 4.2

genaue_kommazahl :: Double
genaue_kommazahl = 42.4242

zeichen :: Char
zeichen = 'z'
```

- Datentypen groß
- Funktionen klein

Typdefinitionen

- bekannter Datentyp -> neuer Datentyp
- type Typsynonym, data neuer algebraischer Datentyp
- Beispiel 1: immutable, Tupel

```
type Datum = (Int,Int,Integer)
```

- Beispiel 2: mutable, Listen

```
type String = [Char]
```

- Beispiel 3: Aufzählung

```
data Obst = Apfel | Birne | Orange | Banane
```

- allgemein zur Typabfrage:
  `ghci> :type name / :t name`

Crashkurs Haskell

1. GHCi Umgebung
2. Konstanten und Datentypen
3. Funktionen <-
4. Fallunterscheidung
5. Rekursion

Primitive Operatoren

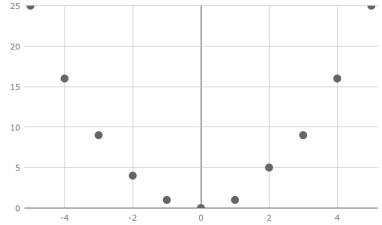
`+ - * / `div` `mod``	für Zahlen
`not || &&`	für Bools
`== /= < > <= >=`	für alles Vergleichbares/Geordnetes

Präfix-/Infixnotation

- Operator **vor** oder **zwischen** Operanden

```
ghci> 3 + 4 == (+) 3 4
True
ghci> 4 `div` 2 == div 4 2
True
```

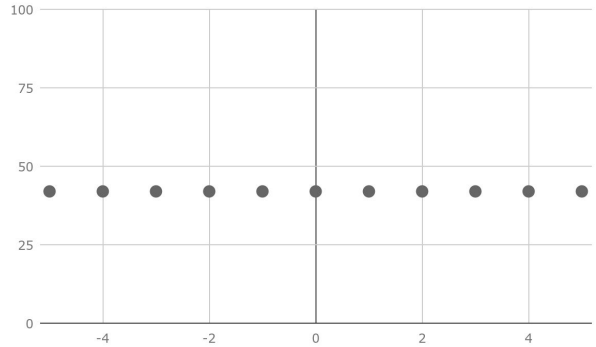
Funktionen




```haskell
quadrat :: Int -> Int
quadrat x = x*x
```


```haskell
implikation :: Bool -> Bool -> Bool
implikation a b = (not a) || b
```

Alles ist Funktion




```
a :: Int
a = 42
```

- auch Konstanten sind Funktionen!

Aufgabe I

Freie Universität Berlin

Schreibt eine Funktion, die:

- Ein Int verdoppelt
- das “Entweder-Oder” darstellt

Aufgabe I

Freie Universität Berlin

Schreibt eine Funktion, die:

- Ein Int verdoppelt
- das “Entweder-Oder” darstellt

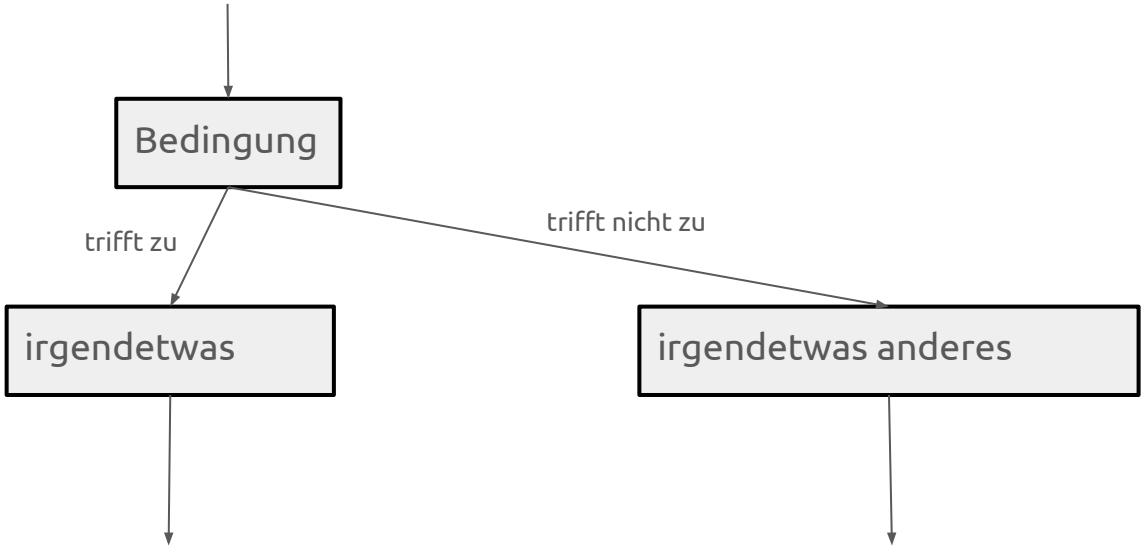
```
verdoppelung :: Int -> Int
verdoppelung x = 2*x
entweder_oder :: Bool -> Bool -> Bool
entweder_oder a b = not (a == b)
```

Crashkurs Haskell

Freie Universität Berlin

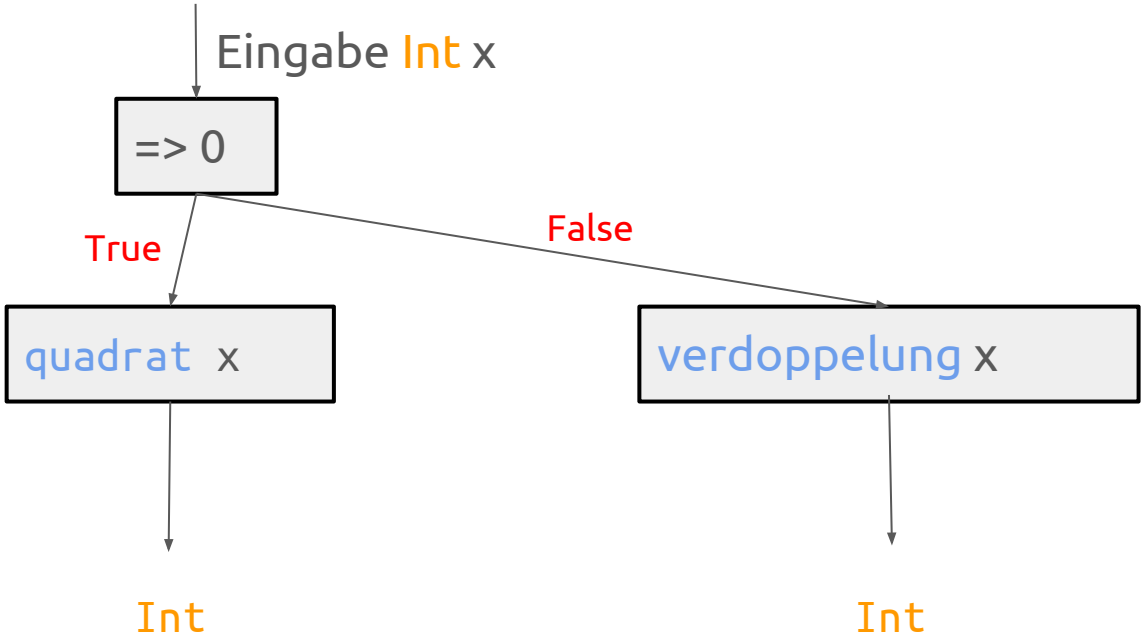
1. GHCi Umgebung
2. Konstanten und Datentypen
3. Funktionen
4. Fallunterscheidung <-
5. Rekursion

Fallunterscheidung

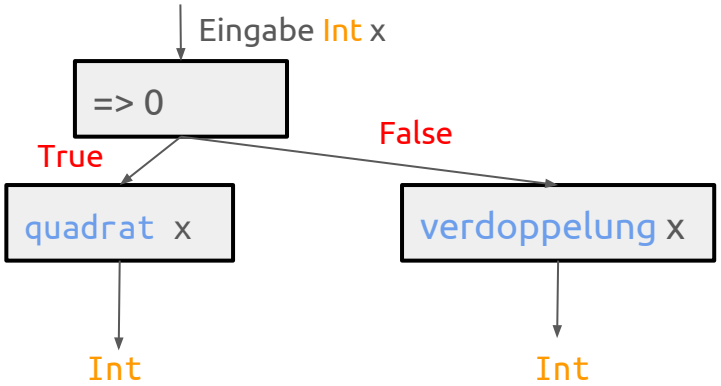


- in Haskell: anderer Wert, selber Typ!

Beispiel 1

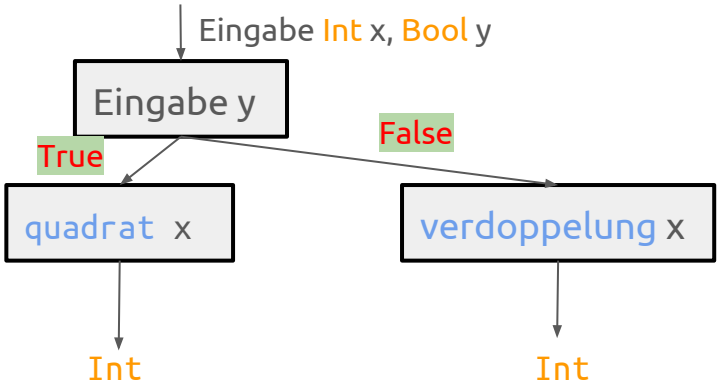


Beispiel 1 - if-then-else



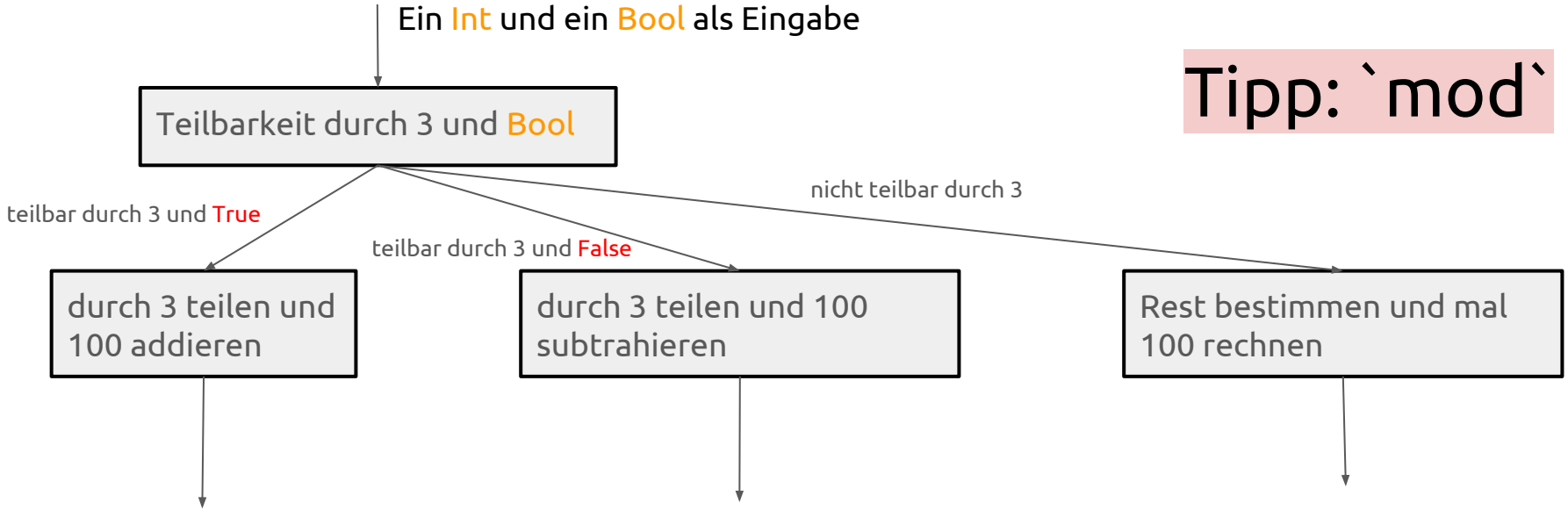

```
quadrat_oder_verdoppelung :: Int -> Int
quadrat_oder_verdoppelung x = if(x >= 0)
                              then quadrat x
                              else verdoppelung x
```

Beispiel 2 - Guards



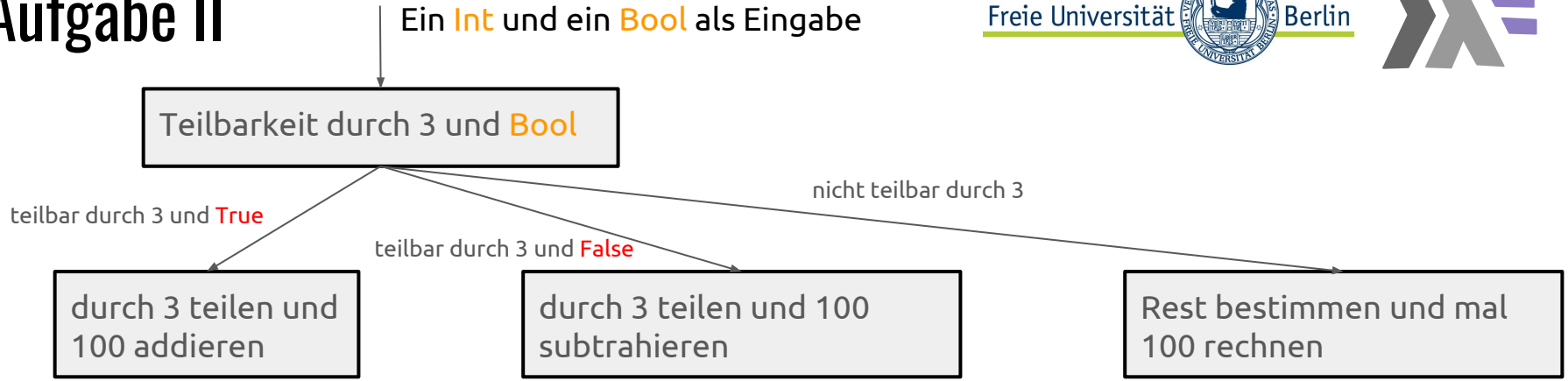

```haskell
quadrat_oder_verdoppelung_2 :: Bool -> Int -> Int
quadrat_oder_verdoppelung_2 y x
    |y = quadrat x
    |otherwise = verdoppelung x
```

Aufgabe II - Guards



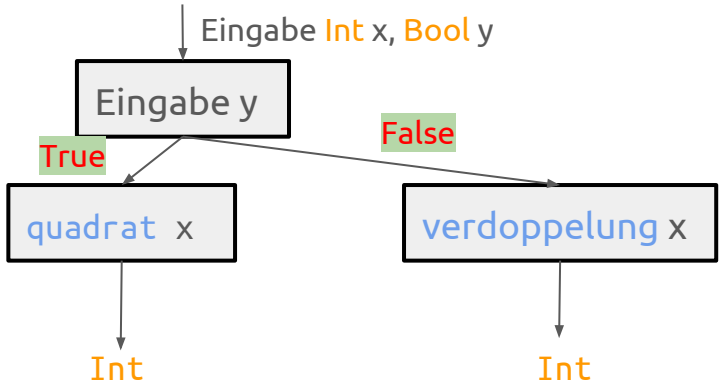
Tipp: `mod`

Aufgabe II



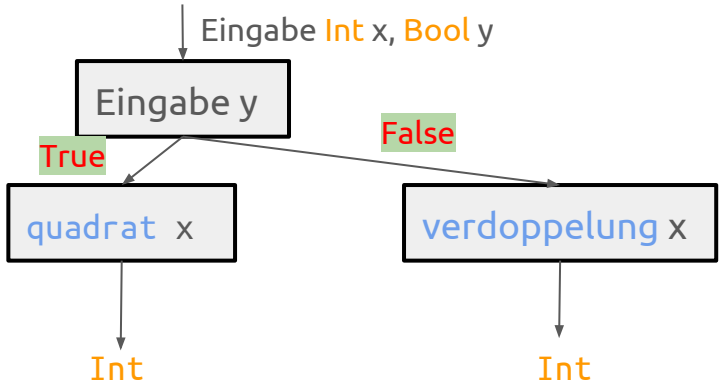

```haskell
teilbar_durch_drei :: Int -> Bool -> Int
teilbar_durch_drei x b
    | (x`mod`3 == 0) && b = x`div`3 + 100
    | (x`mod`3 == 0) && (not b) = x`div`3 - 100
    | otherwise = x`mod`3 * 100
```

Beispiel 2 - Pattern Matching



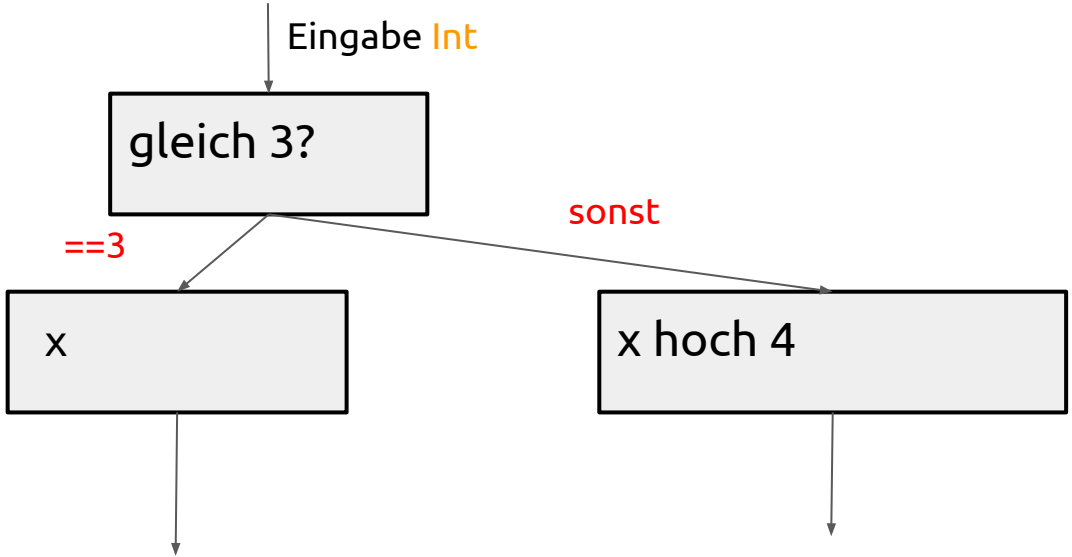

```
quadrat_oder_verdoppelung_3 :: Bool -> Int -> Int
quadrat_oder_verdoppelung_3 True x = quadrat x
quadrat_oder_verdoppelung_3 False x = verdoppelung x
```

Beispiel 2 - Pattern Matching

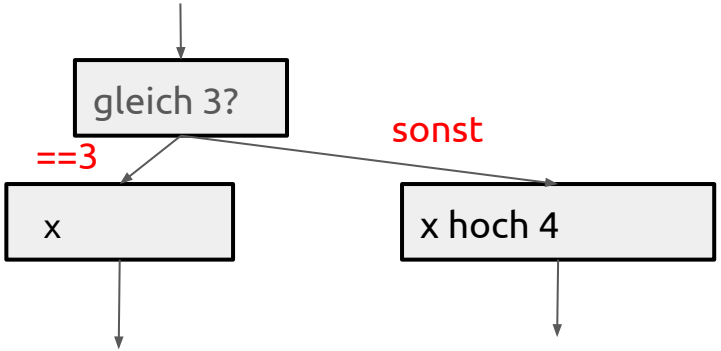



```
quadrat_oder_verdoppelung_3 :: Bool -> Int -> Int
quadrat_oder_verdoppelung_3 True x = quadrat x
quadrat_oder_verdoppelung_3 _ x = verdoppelung x
```

Aufgabe III - Pattern Matching

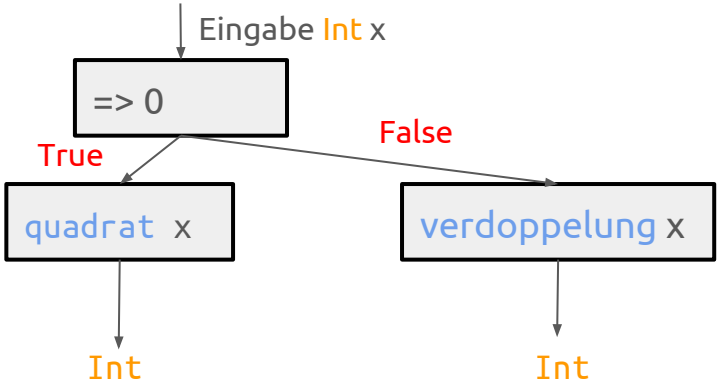


Aufgabe III




```haskell
gleich_drei :: Int -> Int
gleich_drei 3 = 3
gleich_drei x = quadrat (quadrat x)
```

Beispiel 1 - if-then-else



- nicht möglich mit pattern matching!

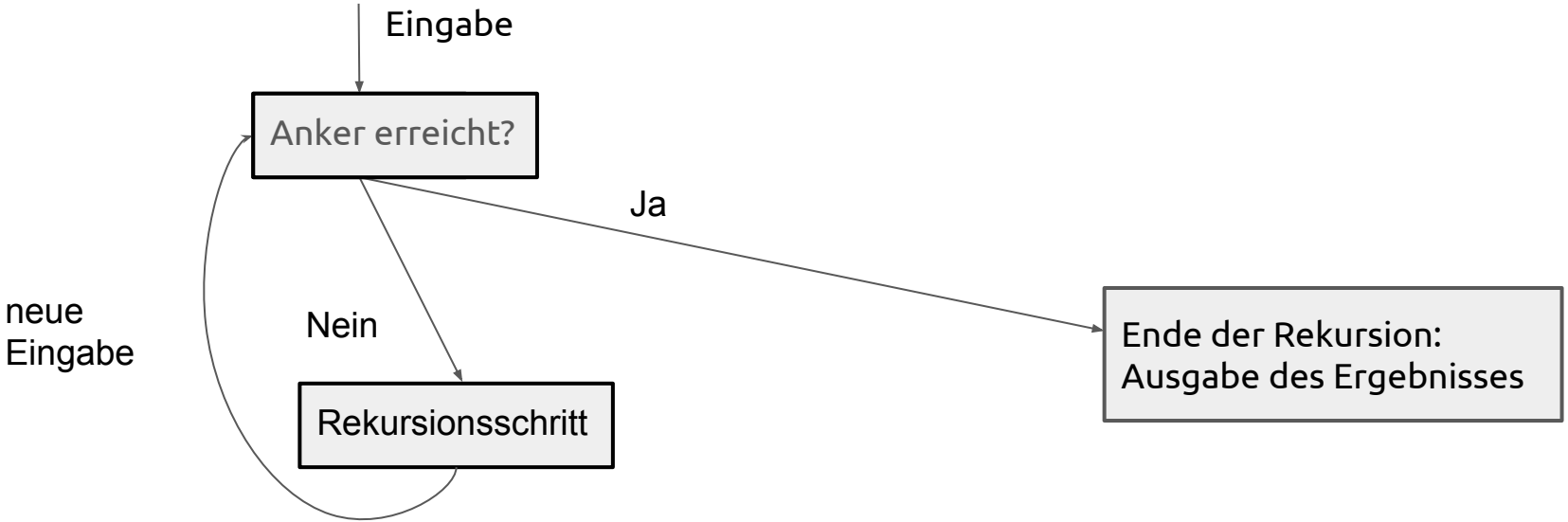
```
quadrat_oder_verdoppelung :: Int -> Int
quadrat_oder_verdoppelung x = if(x => 0)
                              then quadrat x
                              else verdoppelung x
```

Crashkurs Haskell

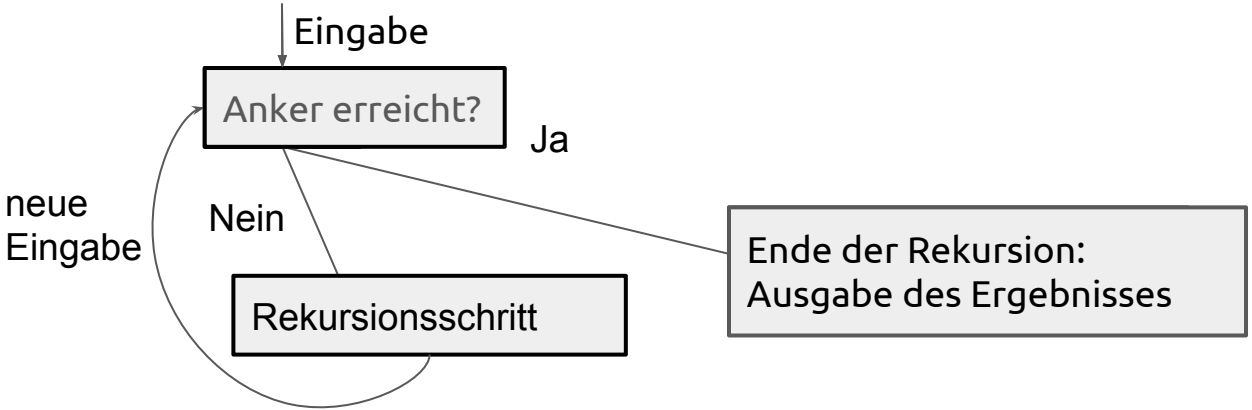
Freie Universität Berlin

1. GHCi Umgebung
2. Konstanten und Datentypen
3. Funktionen
4. Fallunterscheidung
5. Rekursion <-

Rekursion


Eingabe
Anker erreicht?
Ja
neue
Eingabe
Nein
Rekursionsschritt
Ende der Rekursion:
Ausgabe des Ergebnisses

Rekursion




```haskell
fakultaet :: Int -> Int
fakultaet 0 = 1
fakultaet n = n * fakultaet (n-1)
```

Rekursion


```haskell
fakultaet :: Int -> Int
fakultaet 0 = 1
fakultaet n = n * fakultaet (n-1)

-- fakultaet 5
-- 5 * (fakultaet 4)
-- 5 * (4 * (fakultaet 3))
-- 5 * (4 * (3 * (fakultaet 2)))
-- 5 * (4 * (3 * (2 * (fakutaet 1))))
-- 5 * (4 * (3 * (2 * (1))))
```

Rekursion

```
-- fakultaet 5
-- 5 * (fakultaet 4)
-- 5 * (4 * (fakultaet 3))
-- 5 * (4 * (3 * (fakultaet 2)))
-- 5 * (4 * (3 * (2 * (fakutaet 1))))
-- 5 * (4 * (3 * (2 * (1))))
-- 5 * (4 * (3 * (2)))
-- 5 * (4 * (6))
-- 5 * (24)
-- 120
```

Aufgabe IV

Freie Universität Berlin

Schreibt eine Funktion, die

- die n-te Potenz eines Ints berechnet
- die n-te Fibonacci-Zahl berechnet

Aufgabe IV

Freie Universität Berlin

```haskell
potenz :: Int -> Int -> Int
potenz 0 _ = 1
potenz n x = x * (potenz (n-1) x)

fibonacci :: Int -> Int
fibonacci 0 = 0
fibonacci 1 = 1
fibonacci n = (fibonacci (n-1)) + (fibonacci (n-2))
```

Freie Universität Berlin

Vielen Dank

weiterführend: www.learnyouahaskell.com