

Haskell Done Quick (02)

Lars Pfrenger | 22. November 2024

1. Typen und Typklassen

Aber warum?

 "I spent 3 hours trying to find the 'bug' here"

Aber warum?

🐼 "I spent 3 hours trying to find the 'bug' here"

```
elseif args[1] == "SaveToolSlot" then
    local backpackStuff = Leaderstats:Get(player, "Backpack")

    for _,v in pairs(backpackStuff) do
        local match = string.match(v, args[2])

        if match ~= nil then
            backpackStuff[v] = args[2] .. "_" .. args[3]
        end
    end
end
```

Aber warum?

🐞 "I spent 3 hours trying to find the 'bug' here"

```
elseif args[1] == "SaveToolSlot" then
    local backpackStuff = Leaderstats:Get(player, "Backpack")

    for _,v in pairs(backpackStuff) do
        local match = string.match(v, args[2])

        if math ~= nil then
            backpackStuff[v] = args[2] .. "_" .. args[3]
        end
    end
end
```

Aber warum?

- Fehler erkennen bevor das Programm ausgeführt wird
- Kritische Systeme (z.B. Flugzeuge)
- LiquidHaskell → Beweise auf/mit Code führen

Primitive Typen

Type	Wertebereich	Beispiel
Bool	<code>False</code> , <code>True</code>	<code>False</code>
Char	Zeichen	<code>'a'</code>
String	Zeichenkette	<code>"Hallo Welt"</code>
Int	Ganzzahlen mind. $[-2^{29}, 2^{29} - 1]$	42
Integer	Ganzzahlen (theoretisch) beliebig Groß	2^{100}
Float	IEEE single precision Gleitkommazahl	1.2345678
Double	IEEE double precision Gleitkommazahl	1.234567890234567

Warum Typ Klassen?

```
(==) :: Eq => a -> a -> Bool
```


Warum Typ Klassen?

```
(==) :: a -> a -> Bool
```

Warum Typ Klassen?

`(==) :: a -> a -> Bool`

- ⇒ `(==)` würde für alle Typen valide sein
- ⇒ man soll aber nur bestimmte Typen vergleichen können
- ⇒ z.B. Funktionen vergleichen macht kein Sinn

Typklassen Definition

Klassendefinition:

```
class Eq a where  
  (==) :: a -> a -> Bool
```

Typklassen Definition

Klassendefinition:

```
class Eq a where  
    (==) :: a -> a -> Bool
```

Instanz:

```
instance Eq Float where  
    x == y = x `floatEq` y
```

⇒ (==) kann auf `Float` benutzt werden

Typklassen Definition

Klassendefinition:

```
class Eq a where  
    (==) :: a -> a -> Bool
```

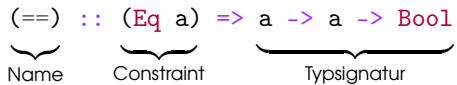
Instanz:

```
instance Eq Float where  
    x == y = x `floatEq` y
```

⇒ (==) kann auf `Float` benutzt werden


Syntax

$(==) :: (Eq\ a) \Rightarrow a \rightarrow a \rightarrow Bool$


Name Constraint Typsignatur

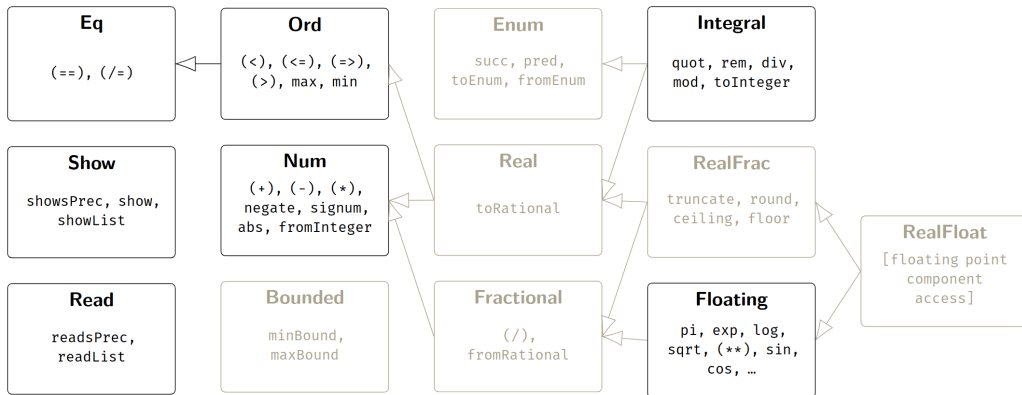
Syntax

$(==) :: (\text{Eq } a) \Rightarrow a \rightarrow a \rightarrow \text{Bool}$



\Rightarrow "Wenn a die Klasse Eq implementiert, dann hat $(==)$ den Typen $a \rightarrow a \rightarrow \text{Bool}$ "

Typklassen



GHCI

```
ghci> :info Num
```

GHCI

```
ghci> :info Num
```

```
type Num :: * -> Constraint
```

```
class Num a where
```

```
    (+) :: a -> a -> a
```

```
    (-) :: a -> a -> a
```

```
    (*) :: a -> a -> a
```

```
    ...
```

```
instance Num Double -- Defined in 'GHC.Float'
```

```
instance Num Float -- Defined in 'GHC.Float'
```

```
instance Num Int -- Defined in 'GHC.Num'
```

```
instance Num Integer -- Defined in 'GHC.Num'
```

```
instance Num Word -- Defined in 'GHC.Num'
```

Ende

Danke für Eure Aufmerksamkeit