Arbeitsblatt: Currying und Partial Application

Jede Funktion in Haskell akzeptiert nur ein Argument. Wenn mehrere Argumente benötigt werden, kann eine Funktion als Resultat eine weitere Funktion zurückgeben, die dann das nächste Argument erwartet. Diesen Mechanismus nennt man Currying und ist benannt nach dem Logiker Haskell B. Curry (1900 – 1982).

Betrachten wir folgendes Beispiel:

```
wrap :: String -> String -> String -> String
wrap pre post text = pre ++ text ++ post
```

Der Pfeil '->' ist rechtsassoziativ, d.h. wenn mehrere Pfeile nacheinander stehen, beginnen Sie rechts die Klammern zu setzen:

```
wrap :: String -> (String -> (String -> String))
```

Jetzt wird es deutlicher. wrap ist eine Funktion, die ein `String` Argument erwartet und als Resultat eine Funktion zurück gibt `(String -> (String -> String))`.

Wozu ist das nützlich? Es erlaubt Funktionen vorzukonfigurieren, indem man nur einen Teil der Parameter übergibt. Zum Beispiel kann man mit wrap eine Funktion (String -> String) erstellen, die den Input String in ein HTML h1 Tag verpackt:

```
wrap "<h1>" "</h1>"
```

Der obige Ausdruck ist dann noch vom Typ String -> String. Das erste Argument vom Typ String und das zweite Argument vom Typ String wurde ja bereits übergeben. Übrig bleibt noch String -> String.

Dieser Funktion kann man jetzt wieder einen Namen geben:

```
h1Tag :: String -> String
h1Tag = wrap "<h1>" "</h1>"
```

Sie kann nun wie folgt verwendet werden:

```
h1Tag "Hi" ~> "<h1>Hi</h1>"
```

Aufgabe: Typen

Gegeben ist die Funktion old:

```
old :: Int -> Bool
old age = age > 90
```

Gesucht sind die Typsignaturen der folgenden Ausdrücke:

`drop 3` :: [a] -> [a]

`map snd` :: [(a,b)] -> [b]

`filter old` :: [Int] -> [Int]

`map head` :: [[a]] -> [a]

`map (wrap "/*" "*/")` :: [String] -> [String]

map map : : [a -> b] -> [[a] -> [b]]

map :: (a -> b) -> [a] -> [b]