Arbeitsblatt: Bedingte Ausführung

Vergleich
Sie haben vier verschiedene Konstrukte kennengelernt um Code nur unter bestimmten Bedingungen auszuführen:

1. Pattern Matching
2. Case expression
3. Guards
4. If-Then-Else-Expression

Dieselbe Funktion kann oft mit mehreren Konstrukten formuliert werden. Betrachten wir eine Funktion die entscheidet ob eine Liste leer ist, ein Element oder mehr als ein Element enthält. Sie lässt sich so formulieren:

Mit Pattern Matching

```
listLength1 :: [a] -> String
listLength1 []  = "empty."
listLength1 [_] = "a singleton list."
listLength1  _  = "a longer list."
```

Mit case expression

```
listLength2a :: [a] -> String
listLength2a xs = case xs of
  []  -> "empty."
  [_] -> "a singleton list."
   _  -> "a longer list."

listLength2b :: [a] -> String
listLength2b xs = case length xs of
  0 -> "empty."
  1 -> "a singleton list."
  _ -> "a longer list."
```

Hier ist ersichtlich, dass die case-Expression mehr Flexibilität anbietet als es bei der Funktionsdefinition der Fall ist. Es ist in der case-Expression nämlich möglich (Variante 2b), einen Ausdruck (`length xs`) zu evaluieren und dessen Ergebnis zu matchen.

Am elegantesten ist die Lösung `listLength1`. Die Variante `listLength2b` ist sehr ineffizient, weil die `length` die ganze Liste durchgehen muss O(n).

Mit guards

```
listLength3 :: [a] -> String
listLength3 xs | length xs == 0 = "empty."
               | length xs == 1 = "a singleton list."
               | otherwise      = "a longer list."
```

Die Schreibweise mit Guards ist sehr kompakt. Allerdings muss hier dieselbe Expression (`length xs`) mehrfach hingeschrieben werden, was die Lesbarkeit etwas einschränkt.

Mit if-then-else

```
listLength4 :: [a] -> String
listLength4 xs = if length xs == 0
                    then "empty."
                    else if length xs == 1
                             then "a singleton list."
                             else "a longer list."
```

if-then-else lässt nur eine Zweiweg-Entscheidung zu. Da wir drei Fälle unterscheiden möchten, sind wir gezwungen zwei if-then-else-Expressions zu verschachteln. Die Lesbarkeit leidet entsprechend.

Aufgabe: Variationen von bedingter Ausführung

Implementieren Sie eine Funktion `max' :: Int -> Int -> Int` welche den grösseren der beiden Inputwerte als Resultat zurückliefert:

```
> max' 3 5
5
> max' (-4) (-8)
-4
```

Implementieren Sie diese Funktion mit Hilfe von

a) Guards
b) if-then-else-Expression
c) case-Expression
d) Können sie die `max'`-Funktion auch mit Hilfe von Pattern-Matching in der Funktionsdefinition implementieren?
   Falls ja: Geben sie eine Implementation an
   Falls nein: Begründen Sie!