Arbeitsblatt: Higher Order Functions II

Auch in diesem Arbeitsblatt geht es um wiederkehrende Rekursionsmuster und wie man vermeiden kann, diese immer wieder von Hand zu implementieren:

Sie haben diverse Funktionen geschrieben, die Elemente von Listen entfernen. Als Beispiel diene die Funktion evens. Sie entfernt aus einer Liste von Ints alle ungeraden Elemente:

```
evens :: [Int] -> [Int]
evens []                = []
evens (i:is) | even i    = i : evens is
             | otherwise = evens is
```

Ein weiteres Beispiel ist die Funktion goodS. Sie nimmt eine Liste von Studierenden und gibt jene mit einer Note grösser 5 als Resultat zurück.

```
data Student = Student { email :: String, grade :: Float }
goodS :: [Student] -> [Student]
goodS [] = []
goodS (s:ss) | grade s > 5 = s : goodS ss
             | otherwise   = goodS ss
```

a) Vergleichen Sie die zwei Funktionen `evens` und `goodS`. Markieren Sie im Code der beiden Funktionen die Gemeinsamkeiten und in welchen Teilen sie sich unterscheiden.
Hinweis: Die Namen der Funktionen sind nicht relevant, es geht um die Struktur der Funktionen.

b) Definieren Sie die Funktion `keep`. Sie soll den Teil, der bei `evens` und `goodS` gleich war implementieren und den Aspekt worin sich die Funktionen unterscheiden als Argument nehmen.
Hinweis: Schreiben Sie zuerst die Typsignatur der Funktion.

c) Implementieren Sie die Funktionen `evens` und `goodS` basierend auf Ihrer neuen Funktion `keep`: