

Funktionale Programmierung

Blatt 2

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 09.10.2018 06:54

Über den Link https://classroom.github.com/a/6ErQRS_n bekommen Sie wieder ein leeres Repository auf GitHub welches Sie analog zu dem für Blatt 1 Aufgabe 3 nutzen können.

Aufgabe 1

Implementieren Sie eine Funktion die **Wortpalindrome** erkennt und entsprechend **True** oder **False** zurück gibt mit folgender Typsignatur:

```
palindrom :: String -> Bool
```

Wenn Sie den eingegebenen **String** bearbeiten wollen, beachten Sie, dass ein **String** eine Liste von **Char** ist. Sie können einen **String** also mit allen Listenfunktionen, wie z.B. **map** oder **filter** bearbeiten.

Informationen über Listenfunktionen finden Sie beispielsweise in der **Prelude**. Weitere Funktionen sind in **Data.List**. Funktionen für einzelne Zeichen finden Sie in **Data.Char**.

Wenn eine verwendete Funktion nicht in der **Prelude** ist, also nicht automatisch importiert wird, müssen Sie sie explizit importieren, z.B.

```
import Data.Char ( toLower )
```

Schreiben Sie Tests für Ihre Funktion. Die HSpec-Dokumentation finden Sie unter <http://hspec.github.io/>.

Aufgabe 2

Implementieren Sie eine Funktion `palindromPhrase` die Satzpalindrome erkennt.

Nutzen Sie zum Schreiben von Tests z.B. die Sätze

- O Genie, der Herr ehre Dein Ego!
- Trug Tim eine so helle Hose nie mit Gurt?

Aufgabe 3

Implementieren Sie folgende Funktion mit Listenkomprehension.

Die Funktion

```
maxPalindrom :: [String] -> Int
```

die aus einer Liste von Zeichenketten die Länge des längsten Palindroms (Wort- oder Satzpalindrom) berechnet.

Bei der Länge des Satzpalindroms zählen Leerzeichen und Satzzeichen mit.

Sie können Ihre Funktion z.B. so testen, dass bei der Liste

```
[ "Otto"  
  , "Anja"  
  , "Eine güldne, gute Tugend: Lüge nie!"  
  , "Blablابلababababsjkd dkjdkjksdsn."  
]
```

35 herauskommen müsste.