

Programmieraufgaben

insgesamt 4 Punkte

Lösen Sie die folgenden Programmieraufgaben. In dem Ordner **Aufgaben** finden Sie neben der Datei **Tests.hs** zwei Haskell-Dateien, welche jeweils den Rumpf mit Typsignatur enthalten. Ergänzen Sie diese mit einer Implementation, und beachten Sie bitte:

- Verändern Sie die vorgegeben Typsignaturen nicht.
- Benennen Sie die Dateien nicht um, und legen Sie keine weiteren Dateien an.
- Achten Sie darauf, dass sich Ihre Abgabe übersetzen lässt; nicht übersetzbare Lösungen werden mit jeweils 0 Punkten bewertet.

Weitere nützliche Hinweise

- Sie können Visual Studio Code zum Bearbeiten der Programmieraufgabe nutzen; es ist mit Erweiterungen für Haskell vorkonfiguriert, und wird durch das Icon auf dem Desktop gestartet.
- Zum Übersetzen benutzen Sie den `ghci`. Auf dem Desktop finden Sie ein Icon, welches den `ghci` im richtigen Verzeichnis startet, so dass Sie mit `:l <Aufgabe>.hs` direkt die Quelldatei laden können.
- Die Datei **Tests.hs** enthält die Beispiele aus der Aufgabenstellung als Unit-Tests. So können Sie schnell prüfen, ob Ihre Lösung plausibel ist. Natürlich sind diese Tests nur notwendig, aber nicht hinreichend für eine vollständige Lösung der Aufgaben.
Zum Ausführen der Tests laden Sie bitte **Tests.hs** in den `ghci`, und rufen die Funktion `run` auf.
- Sie finden auf dem Desktop Icons zur Dokumentation des `ghc` (insbesondere der Standard-Bücherei).

Viel Glück!

1. Histogramme

2 Punkte

Definieren Sie eine Funktion `histogram`, welches ein sehr einfaches Histogramm (eine Häufigkeitsverteilung) zeichnet. `histogram` bekommt ein Füllzeichen und eine Liste von Zahlen, und gibt eine Zeichenkette zurück, die für jede Zahl in der Eingabeliste eine Zeile mit genausoviel Kopien des Füllzeichens beinhaltet. Insgesamt enthält das Ergebnis also genausoviele (mit `\n` getrennte) Zeilen wie die Länge der Eingabeliste.

Beispiel:

```
histogram '*' [3, 5, 2] ~> "***\n*****\n**\n"
```

Wenn wir den String mit `putStr` ausgeben, bewirkt `\n` einen Zeilenumbruch:

```
putStr (histogram '*' [3, 5, 2]) ~>
***
*****
**
```

```
putStr (histogram 'X' [1,5,1,3]) ~>
X
XXXXX
X
XXX
```

Hinweis: Die Funktion `replicate :: Int → a → [a]` kann hilfreich sein.

2. Listen sieben

2 Punkte

Definieren Sie eine Funktion `dropEvery n xs`, welches aus einer Eingabeliste `xs` jedes `n`-te Element entfernt.

Beispiel:

```
dropEvery 3 "abcdefg" ~> "abdeg"
dropEvery 5 [0..14] ~> [0,1,2,3,5,6,7,8,10,11,12,13]
```

Hinweise:

- Wenn Sie Schwierigkeiten haben, einen Ansatz zu finden, implementieren Sie eine rekursive Lösung. Hierfür sind die vordefinierten Funktionen `splitAt :: Int → [α] → ([α], [α])` und `init :: [α] → [α]` hilfreich.
- *Alternativ* können Sie eine Lösung mit Funktionen höherer Ordnung implementieren. Hierzu zerlegen Sie die Aufgabe in drei einfachere Teilschritte:
 1. Erstellen Sie erst eine Funktion, die aus einer Liste `[α]` eine Liste von Paaren `[(Int, α)]` macht, deren erste Komponente der Index in der Liste (angefangen mit 1) ist. Hierbei ist `zip :: [α] → [β] → [(α, β)]` hilfreich.
 2. Erweitern Sie diese Funktion um eine Funktion, die aus einer Liste `[(Int, α)]` alle Paare entfernt, deren erstes Element durch `n` teilbar ist. (Das entspricht den jeweils `n`-ten Elementen der Ursprungsliste.)
 3. Erweitern Sie die Funktion zum Schluss so, dass sie aus einer Liste von Paaren `[(Int, α)]` das jeweils zweite Argument auswählt, und eine Liste `[α]` zurückgibt.

Die vordefinierten Funktionen `fst :: (α, β) → α` und `snd :: (α, β) → β` sind beim Umgang mit Paaren hilfreich.