

- Kreuzen Sie gelöste Aufgaben im OLAT Kurs des Proseminars an.
- Lösen Sie Programmieraufgaben in `Template_05.hs` und laden Sie diese Datei in OLAT hoch.
- Ihre Template-Datei sollte mit `ghci` ohne Fehlermeldung kompilieren.

Aufgabe 1 *Bedingte Gleichungen, Case-Ausdrücke, und lokale Definitionen***5 P.**

- Wir wollen den Binomialkoeffizienten $\binom{n}{k}$ berechnen. Für $0 \leq k \leq n$ haben wir dafür die Formel $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Weil wir die Fakultäten nicht für jeden Aufruf der Funktion neu berechnen wollen, haben wir die Liste `factorials :: [Integer]` vorberechnet, welche die ersten 50 Fakultäten enthält. Implementieren Sie die Funktion `binomial :: Integer -> Integer -> Maybe Integer`. Dabei sollen Sie mit bedingten Gleichungen folgende Fälle unterscheiden:
 - Falls $0 \leq n < k$: Geben Sie `Just 0` zurück.
 - Falls $0 \leq k \leq n$: Versuchen Sie, die oben gegebene Formel mit den Werten aus `factorials` zu berechnen. Falls mindestens einer dieser Werte nicht in `factorials` vorhanden ist, geben Sie `Nothing` zurück. Andernfalls geben Sie `Just  $\binom{n}{k}$` zurück. Da `factorials` in Zukunft erweitert werden könnte, soll nicht die Bedingung $n \leq 50$ geprüft, sondern die Funktion `safeNth :: [a] -> Integer -> Maybe a` und Case-Ausdrücke verwendet werden, um Werte aus `factorials` nachzuschauen.
 - Falls $k < 0$ oder $n < 0$: Geben Sie `Nothing` zurück. (2.5 Punkte)

Hinweis: Sie können zwei `Integer` mit der Funktion `div` dividieren.

Im Rest dieser Aufgabe werden wir schrittweise den Sortieralgorithmus Quicksort¹ implementieren. Die Idee von Quicksort ist es, die zu sortierende Liste anhand eines Pivot-Elements in zwei Teile aufzuteilen: Die Elemente der Liste, die kleiner oder gleich dem Pivot-Element sind, und die Elemente, die größer als das Pivot-Element sind. Dann werden die beiden Teile rekursiv mit Quicksort sortiert, und schließlich die sortierten Teillisten wieder zusammengefügt.

- Implementieren Sie eine Funktion `partition :: Ord a => a -> [a] -> ([a], [a])`, welche ein Pivot-Element `x` und eine Liste `ys` nimmt, und das Paar `(zs1, zs2)` zurückgibt. Dabei soll `zs1` aus den Elementen von `ys` bestehen die kleiner-gleich `x` sind, und `zs2` aus denen die größer als `x` sind. Verwenden Sie bedingte Gleichungen (guards) um zu entscheiden, in welche Teilliste ein Element gehört. (1.5 Punkte)
Hinweis: Sie können lokale Definitionen oder einen Case-Ausdruck verwenden, um auf die Elemente des durch die rekursiven Aufrufe erzeugten Paares zuzugreifen.
- Verwenden Sie `partition`, um eine Funktion `quicksort :: Ord a => [a] -> [a]` zu implementieren, welche eine Liste `xs` nimmt, und diese sortiert. Falls `xs` höchstens ein Element hat, wird `xs` zurückgegeben. Andernfalls ist das erste Element von `xs` das Pivot-Element, mit welchem der Rest der Liste aufgeteilt wird. Diese Teile sollen dann rekursiv mit `quicksort` sortiert und anschließend zusammengefügt werden. (1 Punkt)

¹Siehe <https://de.wikipedia.org/wiki/Quicksort>.

Aufgabe 2 Die Pell-Folge

1 P.

Die Pell-Folge ist eine mathematische Folge, die unter anderem zur Approximation der Wurzel von 2 verwendet werden kann. Sie ist folgendermaßen definiert:

$$P(n) = \begin{cases} 0 & \text{wenn } n = 0, \\ 1 & \text{wenn } n = 1, \\ 2P(n-1) + P(n-2) & \text{sonst.} \end{cases}$$

Die Formel zur Approximation von $\sqrt{2}$ mithilfe der n -ten Zahl der Pell-Folge lautet $(P_{n-1} + P_n)/P_n$.

Implementieren Sie die Funktion `pell :: Int -> Double`, welche die n -te Pell-Zahl berechnet. Verwenden Sie diese, um mit der Funktion `approxSqrt2 :: Int -> Double` $\sqrt{2}$ mithilfe der n -ten Zahl der Pell-Folge zu approximieren.

Aufgabe 3 Rekursion auf Zahlen und Listen

4 P.

1. Implementieren Sie eine Funktion `ranges :: Int -> Int -> [[Int]]`, wobei `ranges l u` als Rückgabewert eine Liste hat, die alle bei `l` startenden und maximal bis `u` reichenden Zahlenbereiche enthält. Überlegen Sie sich auch, wie Sie damit umgehen wollen, wenn `u` kleiner als `l` ist. (2 Punkte)

Beispiele:

- `ranges 1 3 = [[1], [1, 2], [1, 2, 3]]`
- `ranges 6 7 = [[6], [6, 7]]`
- `ranges 4 7 = [[4], [4, 5], [4, 5, 6], [4, 5, 6, 7]]`

2. Implementieren Sie eine Funktion `thresholdAB :: Int -> String -> Char`. Der Wert `thresholdAB n s` soll angeben, ob in `s` der Buchstabe 'A' oder der Buchstabe 'B' zuerst `n` mal vorkommt (wenn man `s` von links nach rechts durchsucht), und dann das entsprechende Zeichen zurückgeben. Kommt keines von beiden `n` mal vor, so soll der Rückgabewert 'C' sein. (2 Punkte)

Beispiele:

- `thresholdAB 3 "CGABBABAAA" = 'B'`, da das Anfangsstück "CGABBAB" dreimal ein 'B' enthält, aber nur zweimal ein 'A'.
- `thresholdAB 5 "AAAA" = 'C'`