RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe A

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`summe xs`

Gibt die Summe aller Elemente der Liste `xs` zurück. Die leere Liste liefert dabei das Ergebnis 0.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe B

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`produkt xs`

Gibt das Produkt aller Elemente der Liste `xs` zurück. Die leere Liste liefert dabei das Ergebnis 1.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe C

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`anzahl xs`

Gibt die Anzahl der Elemente der Liste `xs` zurück.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe D

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`addiere x xs`

Gibt eine Liste zurück, welche die Elemente aus der Liste `xs` jeweils um `x` erhöht enthält.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe E

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`multipliziere x xs`

Gibt eine Liste zurück, welche die Elemente aus der Liste `xs` jeweils mit `x` multipliziert enthält.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe A

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`summe xs`

Gibt die Summe aller Elemente der Liste `xs` zurück. Die leere Liste liefert dabei das Ergebnis 0.

Lösungsvorschlag:

```
summe :: [Int] -> Int
summe []  = 0
summe (x:xs) = x + (summe xs)
```

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe B

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`produkt xs`

Gibt das Produkt aller Elemente der Liste `xs` zurück. Die leere Liste liefert dabei das Ergebnis 1.

Lösungsvorschlag:

```
produkt :: [Int] -> Int
produkt [] = 1
produkt (x:xs) = x * (produkt xs)
```

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe C

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`anzahl xs`

Gibt die Anzahl der Elemente der Liste `xs` zurück.

Lösungsvorschlag:

```
anzahl :: [a] -> Int
anzahl []  = 0
anzahl (x:xs) = 1 + (anzahl xs)
```

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe D

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`addiere x xs`

Gibt eine Liste zurück, welche die Elemente aus der Liste `xs` jeweils um `x` erhöht enthält.

Lösungsvorschlag:

```
addiere :: Int -> [Int] -> [Int]
addiere _ [] = []
addiere x (y:ys) = (x + y) : (addiere x ys)
```

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 10 Gruppe E

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 10 (10 Punkte)

Implementieren Sie die folgende Funktion in Haskell. Geben Sie auch die Typdeklaration an. Verwenden Sie außer Listenkonstruktoren `[]` und `:` (und deren Kurzschreibweise), Vergleichsoperatoren wie `<=`, `==`,... und arithmetischen Operatoren **keine** vordefinierten Funktionen.

`multipliziere x xs`

Gibt eine Liste zurück, welche die Elemente aus der Liste `xs` jeweils mit `x` multipliziert enthält.

Lösungsvorschlag:

```
multipliziere :: Int -> [Int] -> [Int]
multipliziere _ [] = []
multipliziere x (y:ys) = (x * y) : (multipliziere x ys)
```