

## Übung: Rekursion

### 1. Aufgabe: Maximum

a) Implementieren Sie die Funktion `maxl`, die das grösste Element einer Liste findet.  
Hinweis: Definieren Sie zuerst eine Funktion `max'` die zwei Argumente vergleicht und das grössere Argument zurückgibt.

```
maxl :: (Ord a) => [a] -> a
```

b) Evaluieren Sie schrittweise `maxl [2,5,1]`

### 2. Aufgabe: reverse'

a) Implementieren Sie die Funktion, die eine Liste umkehrt.

```
reverse' :: [a] -> [a]
```

b) Evaluieren Sie schrittweise `reverse' [1,2,3]`

### 3. Aufgabe: alternate

Implementieren Sie die Funktion `alternate`. Sie nimmt zwei Listen und gibt eine Liste zurück die abwechselnd von der ersten Liste und der zweiten Liste die Elemente enthält.  
Hinweis: Überlegen Sie sich wie immer zuerst den Typ der gesuchten Funktion.

Hier einige Beispiele:

```
alternate [1,2,3] [4,5,6] ~> [1,4,2,5,3,6]
alternate [1] [4,5,6]     ~> [1,4,5,6]
alternate [1,2,3] [4]     ~> [1,4,2,3]
alternate [] [20,30]      ~> [20,30]
```

### 4. Aufgabe: DIY

Denken Sie sich selbst eine Aufgabe aus. Beschreiben Sie eine Funktion, die man rekursiv implementieren kann und geben Sie die Musterlösung dazu. Und publizieren Sie bitte Ihre Idee auf Piazza.

Hinweis: Wenn Sie Glück haben, wird das vielleicht sogar eine Prüfungsaufgabe.

## Rekursive Grafiken

In dieser Aufgabe programmieren Sie Diagramme. Wir verwenden dazu die Haskell Bibliothek namens *diagrams* (<http://projects.haskell.org/diagrams/>).

### 1) Installation

Kopieren Sie den Ordner 'recdiagrams' auf Ihren Computer und wechseln Sie mit einer Command Shell in dieses Verzeichnis. Danach führen Sie folgendes Kommando aus:

```
> cabal run
```

Bei der ersten Ausführung dieses Kommandos, werden alle benötigten Abhängigkeiten heruntergeladen und kompiliert. Das dauert eine Weile. Aber nur bei der ersten Ausführung. Danach wird das Programm Main.hs ausgeführt.

Falls die Installation nicht funktioniert, können Sie diesen Teil der Übung ignorieren.

### 2) Ausprobieren

Im Projektordner finden Sie das Haskell Script namens Main.hs. Hier sehen Sie dessen Inhalt:

```
{-# LANGUAGE NoMonomorphismRestriction #-}
import System.Environment (withArgs)
import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine

main = withArgs (words "-o diagram.svg -w 400") (defaultMain diag)
  where diag = circle 1 :: Diagram B
```

Wenn Sie das Programm ändern, können Sie es wiederum neu kompilieren und ausführen:

```
> cabal run
```

Dabei wird die main Funktion ausgeführt, und die Datei *diagram.svg* wird geschrieben (-o diagram.svg). SVG ist ein Format zur Speicherung von Vektorgrafik und kann in jedem modernen Web-Browser angezeigt werden. Die generierte Grafik hat eine Seitenlänge von 400 Pixel (-w 400).

Das Bild beinhaltet nur einen Kreis mit dem Radius 1 (*circle 1*). Alle Grössenangaben sind immer relativ. Wenn Sie einen weiteren Kreis erzeugen mit *circle 2* hat dieser einen doppelt so grossen Radius wie der erste. Erst die Outputgrösse legt dann schlussendlich fest, wie viele Pixel das resultierende Bild einnimmt.

Statt Kreise können Sie auch Quadrate (*square*) und Dreiecke (*eqTriangle*) zeichnen. Wenn Sie eine Figur ohne weitere Attribute zeichnen, ist die Fläche der Figur transparent und der Rahmen ist eine schwarze Linie.

Sie können aber die Figuren auch einfärben mit *fc* (fill color):

```
fc black (circle 1)
```

Die Funktion *fc* nimmt eine Farbe und ein Diagramm und gibt das eingefärbte Diagramm zurück.

Um Diagramme zu kombinieren stehen folgende Funktionen/Operatoren zur Verfügung:

*d1 `atop` d2* legt zwei Diagramme übereinander:

```
fc black (square 1) `atop` circle 1
```



*d1 ||| d2* positioniert zwei Diagramme nebeneinander:

```
fc black (square 1) ||| circle 1
```



*d1 === d2* positioniert zwei Diagramme übereinander:

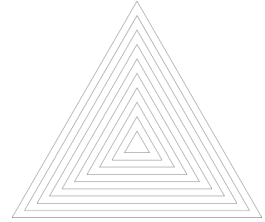
```
fc black (square 1) === circle 1
```



### 3) Aufgaben

**a)** Implementieren Sie eine rekursive Funktion, die  $n$  verschachtelte, gleichseitige Dreiecke zeichnet. Rechts sehen Sie ein Beispiel mit zehn Dreiecken.

Hinweise: Das leere Diagramm heisst mempty



**b)** Basierend auf der Funktion aus Aufgabe a) zeichnen Sie statt Dreiecke Kreise. Zeichnen Sie die Kreise abwechselungsweise schwarz (black) und weiss (white). Rechts sehen Sie ein Beispiel mit 10 Kreisen.

Hinweis: Malen Sie die kleineren Kreise über die grösseren Kreise.



**c)** Zum Schluss generieren wir noch moderne Kunst. Rechts sehen Sie 10 konzentrische Quadrate. Jedes Quadrat ist um  $60^\circ$  gedreht.

Hinweise:

- 1) Verwenden Sie als Basis die Lösung von Aufgabe b)
- 2) Ein Diagramm  $d$  können Sie mit `rotateBy (1/6) d` um einen sechstel Kreis ( $360^\circ/6 = 60^\circ$ ) drehen



**d)** (Optional) Machen Sie was Schönes und präsentieren Sie die Grafik auf Piazza.

Hinweis: Weitere Infos finden Sie unter <http://projects.haskell.org/diagrams/doc/quickstart.html>