Arbeitsblatt: Tuples

Tuples vs Classes

Eine Person könnte in Haskell wie folgt modelliert werden:

```
type Person = (String, String, Int)
```

In Java würde man eine analoge Klasse so definieren:

```
class Person {
  String name;
  String phone;
  int age;
}
```

Eine Klasse welche drei Attribute hat. In Java müssen den Attributen jeweils Namen gegeben werden, in Haskell sind die Komponenten eines Tupels anonym.
Wir haben die Selektoren fst und snd kennengelernt. Sie funktionieren aber nur auf Paaren, also mit 2-Tuplen.
Ihre Definition ist sehr simpel:

```
fst (a, b) = a
snd (a, b) = b
```

Aufgabe 1: Zugriff auf ein bestimmtes Element

Wenn wir in unserem Person Tripel (ein Tupel mit 3 Komponenten) eine spezifische Komponente selektieren wollen, so können wir eine Funktion schreiben, die genau das tut:

```
name :: Person -> String
name (n, p, a) = n
```

Analog zu den Funktionen fst und snd definieren wir also ein Funktion name, welche die erste Komponente eines Person-Tupels ausliest. Präziser: Es ist eine Funktion welche ein bestimmtes Muster (ein Tripel) erkennt und dann in der Lage ist, einen Teil des erkannten Musters zurückzugeben. Mustererkennung heisst im Englischen „Pattern Matching“. Wir werden davon in den nächsten Wochen noch mehrere und komplexere Beispiele sehen.

Nun können wir die Funktion auch nutzen:

```
teacher :: Person
teacher = ("Daniel Kröni", "056 202 78 17", 35)
```

a) Kopieren Sie das Haskell-File tuples.hs vom Active Directory in Ihr Arbeitsverzeichnis und schauen Sie es sich an. Es entält obige Deklarationen.
b) Laden Sie das File in GHCi
c) Führen Sie die Funktion name aus, indem Sie den Namen von teacher abfragen. Auf der Konsole sollte analog folgendes erscheinen:

```
> name teacher
Daniel Kröni
```

d) Definieren Sie ein eigenes Tupel `me`, und fragen Sie auch dort den Namen ab.
Hinweis: sie müssen den Code in tuples.hs ergänzen und danach neu in GHCi laden.
e) Definieren Sie die Funktionen `phone` und `age`, welche Analog zu name die zweite bzw. dritte Komponente einer Person zurückgeben.
f) Überprüfen Sie Ihre Funktionen `phone` und `age` indem Sie `phone me` und `age me` ausführen und sich vergewissern, dass die korrekten Werte zurückgegeben werden.

Aufgabe 2 Tupel-Typen

a) Offensichtlich funktionieren die Funktionen `fst` und `snd` nicht mehr auf dem Person-Triple. Warum?

b) Warum funktionieren die Funktionen `fst` und `snd` nicht auf normalen Tripeln, die nicht vom Typ Person sind?

```
fst ("Hans", "0123456789", 23)

<interactive>:18:5:
 Couldn't match expected type `(a0, b0)'
             with actual type `([Char], [Char], t0)'
 In the first argument of `fst', namely `("Hans", "0123456789", 23)'
 In the expression: fst ("Hans", "0123456789", 23)
 In an equation for `it': it = fst ("Hans", "0123456789", 23)
```

c) Schreiben Sie die Funktionen `first`, `second` und `third` welche die erste, zweite und dritte Komponente eines Tripels (a, b, c) zurückgeben. Das Tripel soll *nicht* von einem bestimmten Typ sein.
Hinweis: Beginnen Sie mit der Typdeklaration der Funktion `first` und deklarieren Sie als Parameter den generischen Tupel Typ.