

*Eine Sache lernt man, indem man sie macht.*  
Cesare Pavese (1908-1950)  
italien. Schriftsteller

*Für das Können gibt es nur einen Beweis, das Tun.*  
Marie von Ebner-Eschenbach (1830-1916)  
österr. Schriftstellerin

## 6. Angabe zu Funktionale Programmierung von Fr, 18.11.2022.

Erstabgabe: Fr, 25.11.2022, 12:00 Uhr

Zweitabgabe: Siehe „Hinweise zu Org. u. Ablauf der Übung“ (TUWEL-Kurs)

**Themen:** *Unechte Polymorphie, Überladung, vor- und selbstdefinierte Typklassen, rekursive Funktionen, Funktionen höherer Ordnung, hierarchische Funktionssysteme, Typdeklarationen (Typsynonyme, neue Typen, algebraische Typen), Feldsyntax, Muster*

**Stoffumfang:** *Kapitel 1 bis Kapitel 11, besonders Kapitel 4, 5, 10 und 11.*

- **Teil A, programmiertechnische Aufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- Teil B, Papier- und Bleistiftaufgaben: Entfallen auf Angaben 5 bis 7.
- Teil C, das Sprachduell: Entfällt auf Angaben 5 bis 7.

## Wichtig

1. Befolgen Sie die Anweisungen aus den ‘Lies-mich’-Dateien (s. TUWEL-Kurs) zu den Angaben sorgfältig, um ein reibungsloses Zusammenspiel mit dem Testsystem sicherzustellen. Bei Fragen dazu, stellen Sie diese bitte im TUWEL-Forum zur LVA.
2. Erweitern Sie für die für diese Angabe zu schreibenden Rechenvorschriften die zur Verfügung gestellte Rahmendatei

`Angabe6.hs`

und legen Sie sie für die Abgabe auf oberstem Niveau in Ihrem *home*-Verzeichnis ab. Achten Sie darauf, dass “Gruppe” Leserechte für diese Datei hat. Wenn nicht, setzen Sie diese Leserechte mittels `chmod g+r Angabe6.hs`.

Löschen Sie keinesfalls eine Deklaration aus der Rahmendatei! Auch dann nicht, wenn Sie einige dieser Deklarationen nicht oder nicht vollständig implementieren wollen. Löschen Sie auch nicht die für das Testsystem erforderliche Modul-Anweisung `module Angabe6 where` am Anfang der Rahmendatei.

3. Der Name der Abgabedatei ist für Erst- und Zweitabgabe ident!
4. Benutzen Sie keine selbstdefinierten Module! Wenn Sie (für spätere Angaben) einzelne Rechenvorschriften früherer Lösungen wiederverwenden möchten, kopieren Sie diese bitte in die neue Abgabedatei ein. Eine `import`-Anweisung für selbstdefinierte Module schlägt für die Auswertung durch das Abgabesystem fehl, weil Ihre Modul-Datei, aus der importiert werden soll, vom Testsystem nicht mit abgesammelt wird.
5. Ihre Programmierlösungen werden stets auf der Maschine `g0` mit der dort installierten Version von `GHCi` überprüft. Stellen Sie deshalb sicher, dass sich Ihre Programme (auch) auf der `g0` unter `GHCi` so verhalten, wie von Ihnen gewünscht.
6. Überzeugen Sie sich bei jeder Abgabe davon! Das gilt besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einer anderen Maschine, einer anderen `GHCi`-Version oder/und einem anderen Werkzeug wie etwa Hugs arbeiten!

## A Programmiertechnische Aufgaben (beurteilt, max. 100 Punkte)

Erweitern Sie zur Lösung der programmiertechnischen Aufgaben die Rahmendatei `Angabe6.hs`. Kommentieren Sie die Rechenvorschriften in Ihrem Programm zweckmäßig, aussagekräftig und problemangemessen. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Wertvereinbarungen für konstante Werte (z.B. `pi = 3.14 :: Float`). Versehen Sie alle Funktionen, die Sie zur Lösung der Aufgaben benötigen, mit ihren Typdeklarationen, d.h. geben Sie stets deren syntaktische Signatur (kurz: Signatur), explizit an.

Wir kehren noch einmal zum Suchmaschinenproblem für den günstigen Einkauf von Waschmaschinen, Wäschetrocknern und Wäscheschleudern von Angabe 4 und 5 zurück, modellieren aber wichtige Daten anders als für Angabe 5 wieder durch Funktionen statt durch Listen von Paaren und untersuchen die relativen Vor- und Nachteile dieser beiden Modellierungsmöglichkeiten.

```
type Nat0    = Int      -- Natürliche Zahlen beginnend mit 0
type Nat1    = Int      -- Natürliche Zahlen beginnend mit 1

newtype EUR  = EUR { euro :: Nat1 }

data Skonto  = Kein_Skonto
             | DreiProzent
             | FuenfProzent
             | ZehnProzent

data Waschmaschine    = M1 | M2 | M3 | M4 | M5
data Waeschetrockner  = T1 | T2 | T3 | T4
data Waescheschleuder = S1 | S2 | S3

data Typ = M Waschmaschine
        | T Waeschetrockner
        | S Waescheschleuder

data Quartal          = Q1 | Q2 | Q3 | Q4 deriving (Eq,Ord,Show)
data Jahr              = J2023 | J2024 | J2025
data Lieferfenster    = LF { quartal :: Quartal,
                             jahr     :: Jahr
                           }

newtype Lieferausblick = LA (Lieferfenster -> Nat0)

data Datensatz
  = DS { preis_in_euro :: Nat1,
        sofort_lieferbare_stueckzahl :: Nat0,
        lieferbare_stueckzahl_im_Zeitfenster :: Lieferausblick,
        skonto :: Skonto
      }
  | Nicht_im_Sortiment

newtype Sortiment = Sort (Typ -> Datensatz)
```

```

data Haendler = H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10

newtype Anbieter = A (Haendler -> Sortiment)

type Suchanfrage = Typ

class Wgf a where
  -- Wgf fuer 'wohlgeformt'
  ist_wgf      :: a -> Bool      -- ist_wgf fuer 'ist wohlgeformt'
  ist_nwgf     :: a -> Bool      -- ist_nwgf fuer 'ist nicht wohlgeformt'
  wgf_fehler   :: a -> a
  -- Protoimplementierungen
  ist_wgf x    = not (ist_nwgf x)
  ist_nwgf x   = not (ist_wgf x)
  wgf_fehler = \x -> error "Argument fehlerhaft"

```

A.1 Für  $M, N$  nichtleere Mengen und  $f : M \rightarrow N$  eine Abbildung von  $M$  nach  $N$  ist der *Wertegraph*  $wg$  von  $f$  eine Teilmenge des kartesischen Produkts  $M \times N$  definiert durch:

$$wg(f) =_{df} \{(m, n) \mid m \in M \wedge n = f(m)\} \subseteq M \times N$$

In Haskell können wir den Wertegraph einer Funktion durch eine Liste seiner Elemente modellieren.

Schreiben Sie Haskell-Rechenvorschriften:

- (a) `wg_la :: Lieferausblick -> [(Lieferfenster, Nat0)]`
- (b) `wg_so :: Sortiment -> [(Typ, Datensatz)]`
- (c) `wg_ab :: Anbieter -> [(Haendler, Sortiment)]`

die jeweils den Wertegraph der Funktion im Datenfeld des Argumentwerts liefern. Die Reihenfolge der Elemente in den Listen spielt keine Rolle; wie die con ihnen modellierten Mengen müssen sie aber duplikatfrei sein.

A.2 Machen Sie die Typen:

- (a) `Lieferausblick`
- (b) `Sortiment`
- (c) `Anbieter`

zu Instanzen der Typklasse `Wgf`. Dabei gilt:

- Ein Wert vom Typ
  - `Lieferausblick` ist wohlgeformt gdw. der Wertegraph der Datenfeldfunktion
  - `Sortiment` ist wohlgeformt gdw. der Wertegraph der Datenfeldfunktion
  - `Anbieter` ist wohlgeformt gdw. der Wertegraph der Datenfeldfunktion
 ist wohlgeformt i.S.v. Angabe 5.
- Die Funktion `wgf_fehler` führt für alle Werte vom Typ
  - `Lieferausblick` zum Aufruf von `error "Ausblickfehler"`.
  - `Sortiment` zum Aufruf von `error "Sortimentfehler"`.

- Anbieter zum Aufruf von `error "Anbieterfehler"`.

Nutzen Sie, wo möglich, die Protoimplementierungen in `Wgf` aus.

**A.3 Ohne Abgabe, ohne Beurteilung:** Im Vergleich zu Angabe 5 haben wir die Deklaration des Typs `Jahr` von `type Jahr = Nat2023` zu `data Jahr = J2023 | J2024 | J2025` verändert. Warum? Auf welche Schwierigkeit wären wir bei der Lösung von Aufgabe A.1 mit der ursprünglichen Deklaration von `Jahr` gestoßen?

**A.4 Ohne Abgabe, ohne Beurteilung:** Wie haben Sie die Funktionen für die Instanzbildungen in Aufgabe A.2 implementiert? Mithilfe der Funktionen aus Aufgabe A.1 und passender Funktionen, die Sie für Angabe 5 geschrieben haben? Wären die Instanzbildungen auch anders möglich? Einfacher? Ohne diesen Rückgriff? Kann sogar auf die Änderung der Deklaration von `Jahr` verzichtet werden?

**A.5** Welche Händler können sofort liefern?

```
type Haendlerliste = [Haendler]

sofort_lieferfaehig :: Suchanfrage -> Anbieter -> Haendlerliste
```

Ist der Wert `w` des Anbieterarguments nicht wohlgeformt, liefert die Auswertung den Wert `wgf_fehler w "Anbieterfehler"`. Ansonsten: kann mehr als ein Händler sofort liefern, soll die Ergebnisliste absteigend sortiert sein (dabei gilt:  $H_1 < H_2 < H_3 < \dots < H_{10}$ ); kann kein Händler sofort liefern, bleibt die Ergebnisliste leer.

**A.6** Wieviel Stück sind sofort erhältlich, wenn bei allen Händlern die jeweils maximal mögliche sofort lieferbare Stückzahl bestellt wird? Wie hoch ist der zugehörige unskontierte Gesamtpreis?

```
type Stueckzahl = Nat0
type Gesamtpreis = Nat0

sofort_erhaeltliche_Stueckzahl :: Suchanfrage -> Anbieter
                                -> (Stueckzahl, Gesamtpreis)
```

Ist der Wert `w` des Anbieterarguments nicht wohlgeformt, liefert die Auswertung den Wert `error "Anbieterargumentfehler"`. Ansonsten: kann kein Händler sofort liefern, weist das Ergebnis jeweils 0 für Stückzahl und Gesamtpreis aus.

**A.7** Welche Händler können im angegebenen Lieferfenster ohne Berücksichtigung eines möglichen Skontorabatts am günstigsten liefern?

```
type Preis = EUR

guenstigste_Lieferanten :: Suchanfrage -> Lieferfenster -> Anbieter
                                -> Maybe Haendlerliste
```

Ist der Wert `w` des Anbieterarguments nicht wohlgeformt, liefert die Auswertung den Wert `error w "Anbieterargumentfehler"`. Ansonsten: kann mehr als ein Händler im angegebenen Lieferfenster günstigst liefern, soll die Ergebnisliste absteigend sortiert (s. A.5) als `Just`-Wert des Typs `Maybe` geliefert werden; kann kein Händler im angegebenen Fenster liefern, wird als Ergebnis der `Maybe`-Wert `Nothing` ausgegeben.

- A.8 Welche Händler können im angegebenen Lieferfenster mindestens die vorgegebene Stückzahl unter Berücksichtigung eines möglichen Skontorabatts am günstigsten liefern? Der rabattierte Preis wird dabei vom unskontierten Gesamtbetrag berechnet und auf den nächsten vollen durch 10 teilbaren Eurobetrag aufgerundet.

```
type RabattierterPreis = EUR
```

```
guenstigste_Lieferanten_im_Lieferfenster ::  
  Suchanfrage -> Lieferfenster -> Stueckzahl -> Anbieter  
  -> [(Haendler,RabattierterPreis)]
```

Ist der Wert `w` des Anbieterarguments nicht wohlgeformt, liefert die Auswertung den Wert `error w "Anbieterargumentfehler"`. Ansonsten: kann mehr als ein Händler im angegebenen Lieferfenster die benötigte Stückzahl günstigst liefern, soll die Ergebnisliste absteigend nach Händlern sortiert sein (s. A.5); kann kein Händler mindestens die benötigte Stückzahl im angegebenen Zeitfenster liefern, wird als Ergebnis die leere Liste geliefert.

- A.9 Ergänzen Sie in den Typdeklarationen dort, wo nötig, **deriving**-Klauseln oder **instance**-Deklarationen, so dass nötige Vergleichstests auf Werten von Typen ausgeführt werden können und alle Ergebnisse am Bildschirm ausgegeben werden können. Dafür sind die Typklassen `Eq`, `Ord`, `Show`, ggf. auch `Enum` wichtig.
- A.10 **Ohne Abgabe, ohne Beurteilung:** In den Aufgaben A.5 bis A.8 haben wir von Angabe 5 übernommen, dass die Auswertung jeweils mit einer Fehlermeldung abbrechen ist, wenn bestimmte Argumentwerte nicht wohlgeformt sind. Ist das schlagend geworden? Hätten wir darauf verzichten können?
- A.11 **Ohne Abgabe, ohne Beurteilung:** Hätten Sie die Aufgaben A.5 bis A.8 auch mit der ursprünglichen Deklaration des Typs `Jahr` mit `type Jahr = Nat2023` lösen können? Begründen Sie Ihre Antwort.
- A.12 **Ohne Abgabe, ohne Beurteilung:** Wenn Sie Ihre Antworten zu A.3, A.4 und A.10, A.11 resümierend betrachten, was spricht für die Modellierung der Typen `Lieferausblick`, `Sortiment`, `Anbieter` durch Listen von Paaren wie auf Angabe 5, was für die Modellierung durch Funktionen wie auf Angabe 6? Wenn Sie die Vor- und Nachteile gegeneinander abwägen, spricht aus Ihrer Sicht mehr für die Modellierung durch Listen oder durch Funktionen? Warum?
- A.13 **Ohne Beurteilung:** Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.
- A.14 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen umfassend mit aussagekräftigen eigenen Testdaten.

## B Papier- und Bleistiftaufgaben

Entfallen auf Angaben 5 bis 7.

## C Das Sprachduell

Entfällt auf Angaben 5 bis 7.

*Iucundi acti labores.*  
*Getane Arbeiten sind angenehm.*  
Cicero (106 - 43 v.Chr.)  
röm. Staatsmann und Schriftsteller