PR3 / Funktionale Programmierung

Übungsaufgaben Nr. 2

1.) Gegeben seien folgende Definitionen:

```
(defun ohne-zahlen (liste)
  (cond ((null liste) nil)
        ((numberp (first liste)) (ohne-zahlen (rest liste)))
        (t (cons (first liste) (ohne-zahlen (rest liste))))))

(defun funktion2 (rest liste)
  (list
   (funcall rest liste)
   (funcall 'rest liste)
   ))
```

Welchen Wert liefert der nachfolgende Ausdruck:

```
(funktion2 'ohne-zahlen '(a 1 b 2 c 3 d 4))
```

Versuchen Sie die Lösung zu finden ohne die Ausdrücke auszuwerten. Ihre Lösung können Sie dann durch Auswerten der Ausdrücke überprüfen. Falls die tatsächliche Lösung von Ihrer Vermutung abweicht, versuchen Sie die Lösung zu verstehen.

2.) Definieren Sie eine Funktion “argument-typ”
Eingabe: arg: ein beliebiger Ausdruck
Wert: Argument-ist-Liste, falls arg eine Liste ist
Argument-ist-Zahl, falls arg eine Zahl ist
Argument-ist-satom, falls arg ein symbolisches Atom ist

3.) Definieren Sie eine Funktion “Skalar-Produkt”, die das Skalarprodukt zweier Vektoren (mit gleicher Dimension: 3 oder beliebig) liefert, d.h. die Funktion soll 2 Parameter haben (die Vektoren), die jeweils als Listen darzustellen sind und eine Zahl als Wert liefern.

Aufrufbeispiel: (skalar-produkt '(2 3 5) '(1 5 4)) --> 37

4.) Definieren Sie eine Funktion “Vektorsumme”, die die Summe zweier Vektoren bildet, also wieder einen Vektor als Wert liefert.

Aufrufbeispiel: (vektorsumme '(2 3 5) '(1 5 4)) --> (3 8 9)

5.) Definieren Sie eine rekursive Funktion “entferne-if”, die eine einstellige Funktion fkt und eine Liste als Argumente hat. entferne-if soll eine Liste liefern, in der alle top-level-Elemente, für die die Anwendung von fkt True liefert, aus der Liste entfernt sind.

Aufrufbeispiel: (entferne-if 'numberp '(a 4 c 7 9 e)) --> (a c e)

6.) Es soll eine rekursive Funktion deep-member geschrieben werden, die einen beliebigen LISP-Ausdruck und eine Liste als Argumente hat. Die Argumente dieser Funktion nennen wir expr und liste. deep-member soll T zurückliefern, falls der Ausdruck expr

irgendwo (in beliebiger Tiefe) in der Liste vorkommt. Wird der Ausdruck expr nirgends in der Liste gefunden, so soll NIL zurückgeliefert werden.

Aufrufbeispiele:
```
(deep-member 'd '(a b (c a d) e))          -->  T
(deep-member '(c a d) '(a (b (c a d)) e))  -->  T
(deep-member 'f '(a b c a d e))            -->  NIL
```

7.) Definieren Sie eine rekursive Funktion "count-numbers", die eine beliebig geschachtelte Liste als Argument hat und zählt, wie viele ihrer Elemente Zahlen sind.

Aufrufbeispiel: `(count-numbers '(a (4 w (4 34) z) 5)) --> 4`

8) Definieren Sie eine Funktion “verteilen”.
Eingabe: liste: eine Liste, die nur Zahlen und Symbole (also keine Unterlisten) enthält.
Wert: eine Liste mit 2 Unterlisten: sliste, zliste
sliste soll alle Symbole von liste enthalten
zliste soll alle Zahlen von liste enthalten.

9.) Definieren Sie eine Funktion “bilde-geordnete-Unterlisten”.
Eingabe: zliste: eine Liste mit Zahlen
Wert: resultliste, eine Liste mit Unterlisten, die jeweils Zahlen enthalten, wobei die Elemente aus zliste, die in streng aufsteigender Reihenfolge vorkommen, jeweils zu einer Unterliste zusammengefasst werden.

Aufrufbeispiel:
```
(bilde-geordnete-Unterlisten `(2 5 9 3 34 44 45 45 56 3 3 5 9 0))
    -->  ((2 5 9) (3 34 44 45) (45 56) (3) (3 5 9) (0))
```