Iteration

1 Typische Iteration (8 Punkte)

Schreibe eine Funktion, die eine Liste als Input nimmt, die verschiedene Objekte enthält. Die Funktion soll die Anzahl der verschiedenen Typen von Objekten zählen, und schön auf dem Bildschirm ausgeben. Benutze dabei entweder `do`, `do*` oder `loop`.

Beispiel:

```
? (count-types (list 1 23 'a "a string" 'symbol
                     '(a list with 6 "different" things)
                     #'+))

== Type == Count ==
  Symbols:   2
  Numbers:   2
  Lists:     1
  Strings:   1
  Unknown:   1

OK
?
```

Hinweis: Benutze die Funktion `format` um das Ergebnis auszugeben. `format` nimmt als erstes Argument einen *stream* – `*standard-output*` ist an den normalen Bildschirm gebunden. Das zweite argument ist ein *string* die u.A. folgende Befehle akzeptiert:

`~%` Newline

`~S` S-expression

Beispiele:

```
? (format *standard-output* "~%; This is a comment.")

; This is a comment.
NIL
? (format *standard-output* "~%; I found ~S numbers!" 42)

; I found 42 numbers!
NIL
? (format *standard-output* "~%; Numbers: ~S~%; Conses: ~S" 42 2)

; Numbers: 42
; Conses: 2
NIL
```

2 Noch mehr Iteration (5 Punkte)

Die Funktionen `!` und `?` sollen jeweils iterativ die Fakultät ihrer Eingabe berechnen; erkläre, warum `!` funktioniert, während `?` für alle Eingaben `0` zurückgibt.

```
(defun ! (n)                (defun ? (n)                    ? (! 7)
  (do ((m n (- m 1))          (do* ((m n (- m 1))           --> 5040
       (k 1 (* k m)))               (k 1 (* k m)))          ? (! 7)
      ((= m 0) k)))                ((= m 0) k)))            --> 0
```

```
? (! 7)
→ 5040
? (? 7)
→ 0
```

3 Vergleichen (8 Punkte)

Schreibe ein Prädikat `my-equal` das feststellt ob zwei S-expressions *isomorph* sind. Achte darauf und *motiviere* warum Du welche Vergleichs-Operatoren verwendet hast.

Hinweis: Es ist nicht erlaubt `equal` zu benutzen. Z.B. wird folgende Lösung mit 0 Punkten beurteilt:

```
(defun my-equal (x1 x2)
  (equal x1 x2))
```

```
:-(
```

Auf Seite 106 in *Steele* (http://www.dfki.de/~becker/lispkurs/clm/indexed-node74.html#Index3) wird eine ähnliche Funktion beschrieben!

4 Schmökaufgabe (3 punkte)

Besuche alle Atome in einem zusammenhängeden Graph exakt einmal. Wenn ihr keine Lösung hinkriegt, skizziert auf jeden Fall eine Lösung – gebt die grundsätzlichen zusätzlichen Probleme an. Ein möglicher Graph wird durch dem folgenden Programm erzeugt:

```
(let ((graph '(a (b nil . nil) c)))
  (setf (second (second graph)) (second graph))
  (setf (rest (last (second graph))) (rest (rest graph)))
  graph)
```

Abgabetermin ist der 17. Dezember 1999