

**Präsenzübungen zur Vorlesung**  
**Deklarative Programmierung: Sommersemester 2018**  
Nr. 7

---

**Aufgabe 7.1: Lagerfeld**




Betrachten Sie die folgenden Implementierungen:

```
(define (draw/circle x y scn)
  (place-image (circle 10 "solid" "red") x y scn))

(define (draw/square x y scn)
  (place-image (square 10 "solid" "blue") x y scn))

(define (draw l scn)
  (match l
    [(cons (struct posn (x y)) xs) (draw xs (draw/square x x
    → scn))]
    [(cons (struct posn (x x)) xs) (draw xs (draw/circle x y
    → scn))]
    [empty scn]))

(draw (list (make-posn 10 10) (make-posn 50 30)
  (make-posn 50 70) (make-posn 90 90))
  (empty-scene 100 100))
```

- a) Beschreiben Sie informell die Funktion des Programms. 
- b) In diesem Programm hat sich ein Fehler eingeschlichen. Beschreiben Sie den Fehler und korrigieren Sie diesen. 
- c) Berechnen Sie schrittweise die Substitution für den match-Aufruf: `(draw (list (make-posn 50 30)) (empty-scene 100 100))`. 

## Aufgabe 7.2: Meinten Sie Rekursion?

Betrachten Sie die folgenden rekursiven Implementierungen:

```
; Number Number -> Number
; check whether b evenly divides a
(define (divisible a b) (equal? (remainder a b) 0))
; Number Number -> Number
; find the greatest common divisor of two numbers
(define (gcd1 a b)
  (local
    [(define (search candidate)
      (cond [(equal? candidate 1) 1]
            [(and (divisible a candidate) (divisible b candidate))
             candidate]
            [else (search (sub1 candidate))]))]
    (search (min a b))))
```




---


```
; Number Number -> Number
; find the greatest common divisor of two numbers
(define (gcd2 a b)
  (local [(define (search larger smaller)
            (cond [(equal? smaller 0) larger]
                  [else (search smaller (remainder larger smaller))]))]
    (search (max a b) (min a b))))
```



a) Erklären Sie die Begriffe "generative Rekursion" und "strukturelle Rekursion". 

b) Wieso terminieren die beiden Programme? 

c) Welches der beiden Programme ist vermutlich schneller? 

Für das generativ rekursive Programm: 

d) Was ist ein trivial lösbares Problem und wie ist dessen Lösung?





e) Wie werden die neuen (leichter zu lösenden) Teilprobleme generiert? Wie viele Teilprobleme werden generiert?

f) Wie wird die Lösung des Problems aus den Teillösungen generiert? Müssen die Teilprobleme zusammengesetzt werden?

### Aufgabe 7.3: Lithium Ionen

Betrachten Sie die folgende Implementierung von Fakultät:

```
(define (fakt/fast n)
  (local
    [(define (f c acc)
      (cond
        [(> c n) acc]
        [else (f (add1 c) (* acc c))])])
    (f 1 1)))
```

- a) Erklären Sie den Begriff Akkumulator-Invariante. 
- b) Geben Sie die Akkumulator-Invariante für f an. 
- c) Geben Sie eine Implementierung von Fakultät ohne Akkumulator an. 
- d) Unter welchen Umständen ist die Einführung eines Akkumulators sinnvoll? 
- e) Lohnt es sich Fakultät mit Akkumulator zu implementieren? Was sind Vor-/Nachteile? 