```
;;;;;;;;;;;;;;;;;
;;
;; teh lisp workshop
;;



;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; part2: paradigmen, bloecke
;;









;;
;; funktionale elemente:
;; funktionen als daten
;;


; funktion auf jedes element einer
; anwenden, aus ergebnissen neue
; liste bauen und zurueckgeben:
(mapcar #'sqrt '(1 4 9 16 25 42))

; weitere funktionen, die funktionen
; als argumente nehmen
(every #'evenp '(2 4 6 8))

(reduce #'+ '(1 2 3 4))
(reduce #'list '(1 2 3 4))









;;
;; lambda
;;

; aufgabe: testen, ob alle zahlen
; einer liste groesser als 10 sind
; mit every

(defun alle-groesser-als-zehn (liste)
  (every (lambda (zahl)
           (> zahl 10))
         liste))
```

```lisp
;;
;; imperative elemente: bloecke
;;

(defun do-lots-of-stuff ()
  (do-something)
  (do-more)
  (do-even-more))

; defun erzeugt einen block. seine
; elemente werden nacheinander evaluiert

; rueckgabewert ist der des letzten elements
; entspricht etwa {} in c/java




; bloecke mit progn
(if t
  (progn
    (format t "foo")
    (format t "bar"))
  (format t "das passiert nicht"))

; da bloecke oft implizit erzeugt werden
; braucht man progn selten.




; bloecke mit gebundenen symbolen mit let
; [defun tut das ja auch]
(let ((foo 23)
      (bar (+ 21 21))
      (format-string "die zahlen sind ~a und ~a.~%"))
```

```
  (format t format-string foo bar)
  (+ foo bar))
```

```
;mit setf werte aendern:
(let ((foo "anfangswert"))
  (format t "wert von foo: ~a~%" foo)
  (setf foo "neuer wert")
  (format t "wert von foo: ~a~%" foo))

; maximum einer liste von zahlen finden:

(defun maximum (liste)
  (let ((max 0))
    (dolist (zahl liste)
      (if (> zahl max)
          (setf max zahl)))
    max))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;    work work!
;;
;; * schreibe funktion, die
;;   * das minimum einer liste von zahlen
;;     zurueckgibt
;;   * eine liste von zahlen normiert
;;     (alle durch den durchschnitt teilt)
;;   * aus einer liste von strings eine neue
;;     liste macht, die alle strings
;;     uppercased, die laenger als 3 chars sind
;;
;; hints: (length '(a b c))    >> 3
;;        (length "foo"        >> 3
;;        (string-upcase "foo" >> "FOO"part
```