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


;;;;;;;;;;;;;;;;;;;;;;;
;;
;; part3: magic
;;









;;
;; macros
;;



;consider this:
(dotimes (index 23)
  (do-some-stuff index))
; warum wird nicht versucht,
; (index 23) auszuwerten?









; dotimes ist ein macro.
; [hier macroexpansion zeigen]


; ein macro sieht aus wie eine funktion
; [steht am anfang einer liste]
; wird aber zur compilezeit evauliert
; gibt neuen code zurueck, der dann anstatt
; des macros eingesetzt wird
; = "es wird expandiert"
```

```lisp
;;
;; macros schreiben
;;

;ein if ohne else-teil
(if something-is-true
  (progn
    (do-stuff)
    (do-more-stuff)))
;sucks.

;wir wollen schreiben:
(when something-is-true
  (do-stuff)
  (do-more-stuff))
; also implizit einen block haben
; ohne extra progn schreiben zu muessen


(defmacro when (condition &rest body)
  `(if ,condition
     (progn ,@body)))



; when ist schon dabei


; macros, die wir kennen:
; when, dotimes, dolist, defun, setf, defmacro



;;
;; the mighty loop macro
;;

(loop for i from 23 to 42 by 2
      do (format t ".")
      collect (* i 2))



; teile des loops:
;
; iteration control:
;   for..from..to, for..in.., etc
; evaluation: do
; combination:
;   collect, count, sum, minimize,...
;


(loop for element in '("foo" "bla" "fasel")
      for i from 1 to 2
      do (format t "index ~a: ~a~%" i element)
```

```
      sum (length element))


;;
;; funcall und apply
;;

(funcall #'+ 1 2 3 4)

(apply #'+ '(1 2 3 4))


(defun repeat-function (fn iterations data)
  (dotimes (i iterations)
    (setf data
          (funcall fn data)))
  data)


(defun repeat-function (fn iterations data)
  (if (= iterations 0)
    data
    (repeat-function fn
                     (- iterations 1)
                     (funcall fn data))))

;zeigen mit sqrt oder sqr




; wenn es mapcar nicht schon gaebe:
(defun mapcar (fn list)
  (loop for element in list
        collect (funcall fn element)))




;;
;;keyword symbols
;;

:foo >> :foo


;verwendung z.b. wie enums:

(defun muss-ich-aufstehen? (wochentag)
  (if (or (eql wochentag :samstag)
          (eql wochentag :sonntag))
    nil
    t))
;verbessern
```

```
;;
;; defparameter
;;
; (defparameter <symbol> <wert>

(defparameter *wochenende*
  '(:samstag :sonntag))

(defun muss-ich-aufstehen? (wochentag)
  (not (member wochentag *wochenende*)))









;;
;; defun reloaded
;;

(defun liste-optional (a &optional b c d)
  (list a b c d))


(defun liste-keys (&key a b c)
  (list a b c))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;   work work!
;;
;; * verbessere den code von vorhin mit loop
;; * schreibe einen funktionsplotter:
;;
;; (plot (lambda (x)
;;         (* x x))
;;       :start -5
;;       :end 5)
;; ........................*
;; ................*
;; ........*
;; ....*
;; .*
;; *
;; .*
;; ....*
;; ........*
;; ................*
;; ........................*
;;
;; (plot #'sqrt)
;; *
;; .*
```

```
;; .*
;; ..*
;; ..*
;; ..*
;; ..*
;; ...*
;; ...*
;; ...*
;; ...*


;loesung coden
```