```
; Ein paar Helper...
;
(require '[clojure.java.io :as io])
(import 'clojure.lang.RT)
(import 'java.io.LineNumberReader)
(import 'java.io.InputStreamReader)
(import 'java.io.PushbackReader)
(import 'java.io.FileReader)

(defn maybe-file-stream
  [path]
  (let [filepath (io/file path)]
    (when (.exists filepath)
      (io/input-stream filepath))))

(defn code*
  [x]
  (when-let [v (resolve x)]
    (when-let [filepath (:file (meta v))]
      (when-let [strm (or (.getResourceAsStream (RT/baseLoader) filepath)
                          (maybe-file-stream filepath))]
        (with-open [rdr (LineNumberReader. (InputStreamReader. strm))]
          (dotimes [_ (dec (:line (meta v)))] (.readLine rdr))
          (let [text (StringBuilder.)
                pbr (proxy [PushbackReader] [rdr]
                      (read [] (let [i (proxy-super read)]
                                 (.append text (char i))
                                 i)))]
            (read (PushbackReader. pbr))
            (str text)))))))

(defmacro code
  [n]
  `(println (or (code* '~n) (str "Source not found"))))

(def wait #(Thread/sleep %))

; Beispiele
;
(def v [1 2 3])
(def l (list 1 2 3))
(def m {:a 1 :b 2 :c 3})
(def s #{1 2 3})
(def schachtel {:a {:b 1}})

(declare defcon rockets)

(defmacro norad
  [ctor]
  `(do
     (def defcon (~ctor 1))
     (def rockets (~ctor :hold))))

(defn start-codes-valid?
  []
  (wait 10000)
  true)

(defn start-rockets-a
  []
  (when (and (= @defcon 1) (start-codes-valid?))
    (swap! defcon dec)
    (reset! rockets :fire)))

(defn stop-doomsday-a
  []
  (reset! defcon 2))

(defn start-rockets-r
  []
  (dosync
    (when (and (= @defcon 1) (start-codes-valid?))
      (alter defcon dec)
      (ref-set rockets :fire))))

(defn stop-doomsday-r
```

```
  []
  (dosync
    (ref-set defcon 2)))

(defn start-rockets-example
  [starter-fn stopper-fn]
  (let [starter (future-call starter-fn)]
    (wait 4)
    (println @defcon @rockets)
    (stopper-fn)
    (println @defcon @rockets)
    @starter
    (println @defcon @rockets)))

(defn clever-map
  [f coll]
  (into (empty coll) (map f coll)))

(defmacro with-file
  [[local file] & body]
  `(let [~local (FileReader. ~file)]
     (try
       ~@body
       (finally
         (.close ~local)))))

nil
```