```lisp
(defparameter *options*
  '("
 aas asd as asdas asd as assaas dasdas
 (c) 2022

 line 1 3wwesas
 line 33323 3242323

 OPTIONS:"
    (cautious  "-c"    "abort on any error    "   t)
    (enough    "-e"    "enough items for a sample"   512)
    (far       "-F"    "far away              "   .9)
    (file      "-f"    "read data from file   "   "../data/auto93.csv")
    (help      "-h"    "show help             "   nil)
    (license   "-l"    "show license          "   nil)
    (e         "-e"    "euclidean coefficient "   2)
    (seed      "-s"    "random number seed    "   10019)
    (todo      "-t"    "start up action       "   "")))

;;;;----------------------------------------------------------------------------
(defun settings (help options)
  (labels ((thing (x) (let ((y (ignore-errors (read-from-string x))))
                        (if (numberp y) y x)))
           (trim (s) (string-trim '(#\Tab #\Space) s))
           (lines (s &optional (c #\,) (n 0) &aux (pos (position c s :start n)))
                  (if pos
                    (cons (subseq s n pos) (lines s c (1+ pos)))
                    (list (subseq s n))))
           (args () #+clisp ext:*args* #+sbcl (cdr sb-ext:*posix-argv*))
           (has (x) (member x (args) :test #'equal))
           (cli (lst &aux (flag (second lst)) (b4 (fourth lst)))
                (list (first lst) flag (third lst)
                      (if (has flag) (cond ((equal b4 t)   nil)
                                           ((equal b4 nil) t)
                                           (t (thing (second (has flag)))))
                        b4))))
    (cons (cons 'about (mapcar #'trim (lines help #\Newline)))
          (mapcar #'cli options))))

(defun print-settings (s &optional (str t))
  (dolist (x (cdar s)) (format str "~&~a~%"   x))
  (dolist (x (cdr s))
    (format str "~& ~a  ~a = ~a" (second x) (third x) (fourth x))))

(setf *options* (settings (car *options*) (cdr *options*)))
(defmacro ? (x) `(third (cdr (assoc ',x (cdr *options*)))))

;;;;----------------------------------------------------------------------------
(defun klassp (x) (eq (charn x) #\!))
(defun lessp  (x) (eq (charn x) #\<))
(defun morep  (x) (eq (charn x) #\>))
(defun goalp  (x) (or (morep x) (lessp x) (klassp x)))
(defun nump   (x) (eq (char0 x) #\.))
(defun char0  (x) (char (symbol-name x) 0))
(defun charn  (x &aux (s (symbol-name x))) (char s (1- (length s))))

(print-settings *options*)

(defmacro deca (x a &optional (inc 1))
  `(decf (cdr (assoc ,x ,a :test #'equal)) ,inc))

(defmacro inca (x a &optional (inc  1))
  `(incf (cdr (or (assoc ,x ,a :test #'equal)
                  (car (setf ,a (cons (cons ,x 0) ,a))))) ,inc))

(defun per (seq &optional (p .5)) (elt seq (floor (* p (length seq)))))
(defun sd  (seq &optional (key #'identity))
  (/ (- (funcall key (per seq .9)) (funcall key (per seq .1))) 2.56))

(defun ent (a &optional (n 0) (e 0))
  (dolist (two a)   (incf n (second two)))
  (dolist (two a e) (let ((p (/ (second two) n))) (decf e (* p (log p 2))))))

(defun csv (file &aux out it)
  (with-open-file (str file)
    (loop (if (setf it (read str nil))
            (push it out)
            (return-from csv (reverse out))))))

(defstruct range lo hi id here n div)

(defun argmin (out xys lo hi b4 here trivial enough)
  (labels ((y (i) (second (elt xys i)))
           (x (i) (first  (elt xys i))))
    (let (lhs rhs cut)
      (loop for i from lo to hi do (inca (y i) rhs))
      (let ((div (ent rhs)))
        (if (> (- hi (1+ lo)) (* 2 enough))
          (loop for i from lo to hi do
                (inca (y i) lhs)
                (deca (y i) rhs)
                (let ((n1 (- i (1+ lo)))
                      (n2 (- hi i)))
                  (if (and (> n1 enough)
                           (> n2 enough)
                           (not (equal (x i) (x (1+ i))))
                           (> (- (x i) (x lo)) trivial)
                           (> (- (x hi) (x i)) trivial))
                    (let ((xpect (/ (+ (* n1 (ent lhs)) (* n2 (ent rhs)))
                                    (+ n1 n2))))
                      (if (< xpect div) (setf cut i
                                              div xpect)))))))
        (if cut
          (setf b4 (argmin out xys lo       cut b4 here trivial enough)
                b4 (argmin out xys (1+ cut) hi b4 here trivial enough))
          (push (make-range :lo b4 :hi (x hi) :id (length out)
                            :here here :n (- hi (1+ lo)) :div div) out)))))
  (range-hi (car out)))

(defun bins (xys where)
  (let* (out
         (n   (length xys))
         (xys (sort (coerce xys 'vector) #'< :key #'first)))
    (argmin out xys 0 (1- n) 1E32 where
            (* (? cohen) (sd xys #'first))
            (floor (/ (length xys) (? bins))))
    (setf (range-hi (elt out (1- n))) most-positive-fixnum)
    out))

(defun make () (load 'bnb))
```