```lisp
;(defpackage :sublime (:use :cl))
;(in-package :sublime)

;;   )\.--.       .-.      /(,-.   .')       .'(   )\    )\   )\.---.
;;  (   ._.'  ,'  /  )  ,' _   )  ( /        \  ) (  ',/ /  (   ,-._(
;;   `-.`.   (  ) | (  (  '-' (   ))        ) (   )    (    \  '-,
;;  ,_ (  \   ) '._\ )  ) _   )  )'._.-.   \  ) (  \(\ \    ) ,-`
;; (  '.)  ) (  ,   (  (  '-' /  (       )  ) \  `.) /  )  (  ``-.
;;  '._,_.'   )/ ._.'   )/._.'    )/,__.'    )/      '.(    )..-.(

;; (quote
;;    (an (elegant (weapon
;;        (for (a (more
;;            (civilized age)))))))

;;                      __  _
;;  __   ___   _ _     / _|(_)  __ _
;; / _| / _ \ | ' \   |  _|| | / _` |
;; \__| \___/ |_||_|  |_|  |_| \__, |
;;                             |___/

(defstruct cli key flag help value)
(defstruct options
  (help
    "sbcl --noinform --script expose.lisp [OPTIONS]
(c) 2022, Tim Menzies, MIT license

Lets have some fun.")
  (options
   (list
    (cli! 'cautious "-c" "about on any error"         t)
    (cli! 'enough   "-e" "enough items for a sample" 512)
    (cli! 'far      "-F" "far away         " .9)
    (cli! 'file     "-f" "read data from file    " "../data/auto93.csv")
    (cli! 'help     "-h" "show help        " nil)
    (cli! 'license  "-l" "show license       " nil)
    (cli! 'p        "-p" "euclidean coefficient   " 2)
    (cli! 'seed     "-s" "random number seed    " 10019)
    (cli! 'todo     "-t" "start up action      " ""))))

(defmethod print-object ((c cli) s)
  (with-slots (key flag help value) c
    (format s "  ~5a ~a " flag  help)
    (if (member value '(t nil)) (terpri s) (format s "= ~a~%" value))))

(defmethod print-object ((o options) s)
  (with-slots (help options) o
    (format s "~a~%~%OPTIONS:~%" help)
    (dolist (x options) (print-object (cdr x) s))))

(defun item (x)
  "Return a number or a trimmed string."
  (cond ((numberp x)   x)
        ((equal x "?") nil)
        (t (let ((y (ignore-errors (read-from-string x))))
             (if (numberp y) y x)))))

(defun cli! (key flag help value)
  (let* ((args (cdr sb-ext:*posix-argv*))
         (it   (member flag args :test #'equal)))
    (if it (setf value (cond ((equal it t)   nil)
                             ((equal it nil) t)
                             (t (item (second it))))))
    (cons key (make-cli :key key :flag flag :help help :value value))))

(defvar *the* (make-options))
```

```lisp

;;  _  _  _
;; | |(_)| |__
;; | || || '_ \
;; |_||_||_.__/

;;; macros
(defmacro $   (x)  `(cli-value (cdr (assoc ',x (options-options *the*)))))
(defmacro aif (? y &optional n) `(let ((it ,?)) (if it ,y ,n)))
(defmacro ?   (p x &rest xs) (if (null xs) `(getf ,p ',x) `(? (getf ,p ',x),@xs)
))

;;; random
(defvar *seed* 10013)
(defun randi (&optional (n 1)) (floor (* n (/ (randf 1000.0) 1000))))
(defun randf (&optional (n 1.0))
  (setf *seed* (mod (* 16807.0d0 *seed*) 2147483647.0d0))
  (* n (- 1.0d0 (/ *seed* 2147483647.0d0))))

;;; lists
(defun nshuffle (lst)
  "Return a new list that randomizes over of lst"
  (let ((tmp (coerce lst 'vector)))
    (loop for i from (length tmp) downto 2
          do (rotatef (elt tmp (random i)) (elt tmp (1- i))))
    (coerce tmp 'list)))

(defun per (lst &optional (p .5)) (elt lst (floor (* p (length lst)))))

;;; defthings
(defmacro defthing (x &rest slots &aux (id (gensym)))
  "Defines structs with uniq ids `_id` and a constuctor `(%make-x)`
 and a print method that hides privates slots (those starting with `_`)."
  (labels ((hidep (z) (equal (char (symbol-name z) 0) #\_))
           (name  (z) (if (consp z) (car z) z))
           (names ()  (remove-if #'hidep (mapcar #'name slots)))
           (%make ()  (intern (format nil "%MAKE-~a" (symbol-name x)))))
    `(let ((,id 0))
       (defstruct (,x (:constructor ,(%make))) (_id (incf ,id)) ,@slots)
       (defmethod print-object ((it ,x) s) (show-object it ',x ',(names) s)))))

(defun show-object (it klass slots s)
  (labels ((show (z)  (let* ((k (intern (symbol-name z) "KEYWORD"))
                             (v (slot-value it z)))
                        (if v `(,k ,v) k))))
    (print-object (cons klass (mapcar #'show slots)) s)))

;;; files
(defmacro with-csv ((lst file &optional out) &body body)
  `(progn (%with-csv ,file (lambda (,lst) ,@body)) ,out))

(defun %csv (file &optional (fn 'print))
  "Run a function 'fn' over file (sub-function of 'with-csv')."
  (with-open-file (str file)
    (loop (funcall fn (or (read-line str nil) (return-from %csv))))))
```

```lisp
;;  _     _      _
;; | |_  | |_  (_)  _ _    __ _   ___
;; |  _| | ' \  | | | ' \  / _` | (_-<
;;  \__| |_||_| |_| |_||_| \__, | /__/
;;                         |___/

(defthing num (at 0) (txt "") (n 0) (w 1) (mu 0) (m2 0) (sd 0) max (ok t)
              (lo most-positive-fixnum) (hi most-negative-fixnum)
              (_has (make-array  32 :fill-pointer 0 :adjustable t)))

(defthing sym    (at 0) (txt "") (n 0) has mode (most 0))
(defthing cols   all x y klass)
(defthing sample rows cols)
(defthing range  col lo hi has)

;;  _ _    _  _   _ __
;; | ' \  | || | | '  \
;; |_||_|  \_,_| |_|_|_|

(labels ((lettern (x &aux (n (length x))) (and (> n 0) (subseq x (- n 1) n))))
  (defun lessp  (x) (equal "-" (lettern x)))
  (defun morep  (x) (equal "+" (lettern x)))
  (defun klassp (x) (equal "!" (lettern x)))
  (defun nump   (x) (upper-case-p (char x 0)))
  (defun goalp  (x) (or (klassp x) (lessp x) (morep x))))

(defun make-num (n &optional (at 0) (txt ""))
  (%make-num :at at :txt txt :max n :w (if (lessp txt) -1 1)))

(defun make-sym (&optional (at 0) (txt ""))
  (%make-sym :at at :txt txt))

(defmethod add ((nu num) x)
  (with-slots (lo hi max ok n _has) nu
    (unless (null x)
      (setf lo (min x lo)
            hi (max x hi)
            n  (1+ n))
      (cond ((> max (length _has))
             (setf ok nil)
             (vector-push-extend x _has))
            ((< (randf) (/ max n))
             (setf ok nil
                   (elt _has (randi (length _has))) x)))))
  x)

(defmethod mid ((n num)) (per (has n) .5))
(defmethod div ((n num)) (/ (- (per (has n) .9) (per (has n) .1)) 2.56))

(defmethod has ((n num))
  (with-slots (ok _has) n
    (unless ok  (setf ok   t
                      _has (sort _has #'<)))
    _has))

;;;; coerce
(Defun str->items (s &optional (c #\,) (n 0) &aux (pos (position c s :start n)))
  "Divide string `s' on character `c'."
  (if pos
      (cons (item (subseq s n pos)) (str->items s (1+ pos)))
      (list (item (subseq s n)))))
```

```lisp
;;  _                _
;; | |_   ___   ___ | |_   ___
;; |  _| / -_) (_-< |  _| (_-<
;;  \__| \___| /__/  \__| /__/

(defvar *tests* nil)

(defmacro deftest (name params  doc  &body body)
  `(progn (pushnew ',name *tests*) (defun ,name ,params ,doc ,@body)))

(defun demos (&optional what quit &aux (fails 0))
  (dolist (one *tests* (if quit (exit :code fails)))
    (let ((doc (documentation one 'function)))
      (when (or (not what) (eql one what))
        (setf *the* (make-options))
        (setf *seed* ($ seed))
        (multiple-value-bind
              (_ err)
            (if ($ cautious)
                (values (funcall one) nil)
                (ignore-errors (funcall one *the*)))
          (identity _)
          (incf fails (if err 1 0))
          (if err
              (format t "~&~&FAIL: [~a] ~a ~a~%" one doc  err)
              (format t "~&~&PASS: [~a] ~a~%"    one doc))))))))


(deftest aa? () "ads" (print 1))
(deftest bb? () "ads" (print 2))

;(defun file2sample (file &aux ((s (make-sample))))
;;;; lib
;;; lists

(defun make () (load "sublime.lisp"))


                                        ; file to samples
                                        ; samples to clusters
                                        ; clusters to ranges
                                        ; ranges to tree
```