```lisp
;;;; vim: ts=2 sw=2 et :
;;;;     __                 __                  __                   __
;;;;    /\ \               /\ \                /\ \                 /\ \
;;;;    \ \ \____   _ __   \ \ \/'\      ___    \ \ \____     __     \_\ \
;;;;     \ \ '__`\ /\`'__\  \ \ , <    /' _ `\   \ \ '__`\  /'__`\   /'_` \
;;;;      \ \ \L\ \\ \ \/    \ \ \\`\  /\ \/\ \   \ \ \L\ \/\ \L\.\_/\ \L\ \
;;;;       \ \_,__/ \ \_\     \ \_\ \_\\ \_\ \_\   \ \_,__/\ \__/.\_\ \___,_\
;;;;        \/___/   \/_/      \/_/\/_/ \/_/\/_/    \/___/  \/__/\/_/\/__,_ /

;;;;      .-------.
;;;;      | Ba    | Bad <----.  planning= (better - bad)
;;;;      |    56 |          |  monitor = (bad - better)
;;;;      .-------.------.   |
;;;;              | Be    |   v
;;;;              |    4  | Better
;;;;              .------.

(defvar *options* '(
  about      "brknbad: explore the world better, explore the world for good.
     (c) 2022, Tim Menzies

     OPTIONS: "
  cautious  ("-c"  "abort on any error       "  t)
  dump      ("-d"  "stack dumps on error     "  nil)
  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)
  p         ("-p"  "euclidean coefficient    "  2)
  seed      ("-s"  "random number seed       "  10019)
  todo      ("-t"  "start up action          "  "nothing")))

;;;; Copyright (c) 2021 Tim Menzies
;;;;
;;;; This is free and unencumbered software released into the public domain.
;;;;
;;;; Anyone is free to copy, modify, publish, use, compile, sell, or
;;;; distribute this software, either in source code form or as a compiled
;;;; binary, for any purpose, commercial or non-commercial, and by any
;;;; means.
;;;;
;;;; In jurisdictions that recognize copyright laws, the author or authors
;;;; of this software dedicate any and all copyright interest in the
;;;; software to the public domain. We make this dedication for the benefit
;;;; of the public at large and to the detriment of our heirs and
;;;; successors. We intend this dedication to be an overt act of
;;;; relinquishment in perpetuity of all present and future rights to this
;;;; software under copyright law.
;;;;
;;;; THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
;;;; EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
;;;; MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
;;;; IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
;;;; OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
;;;; ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
;;;; OTHER DEALINGS IN THE SOFTWARE.
;;;;
;;;; For more information, please refer to <http://unlicense.org/>

;;;    ____ _     ____ ___  ____ _     ____
;;;   | __ |     |  | |__] |__| |     [__
;;;   |__] |___  |__| |__] |  | |___  ___]

(defvar *tests* nil)   ; list of test functions
(defvar *fails* 0)     ; counter for test failires
(defvar *seed* 10019)  ; initial value random nunber seed

;;;    _  _ ____ ____ ____ ____ ____
;;;   |\/| |__| |    |__/ |  | [__
;;;   |  | |  | |___ |  \ |__| ___]

(defmacro ? (x) ;;
  "short hand for access option fields"
  `(third (getf *options* ',x)))

(defmacro o (s x &rest xs)
  "shorthand for recurisve calls to slot-valyes"
  (if xs `(o (slot-value ,s ',x) ,@xs) `(slot-value ,s ',x)))

(defmacro has (x a)
  "ensure `a` has a cells `(x . number)` (where number defaults to 0)"
  `(cdr (or (assoc ,x ,a :test #'equal)
            (car (setf ,a (cons (cons ,x 0) ,a))))))

(defmacro deftest (name params &body body)
  "define a test function"
  `(progn (pushnew ',name *tests*) (defun ,name ,params  ,@body)))

(defmacro with-csv ((lst file &optional out) &body body)
  "file reading iterator"
  (let ((str (gensym)))
    `(let (,lst) (with-open-file (,str ,file)
                   (loop while (setf ,lst (read-line ,str nil)) do ,@body))
       ,out)))

;   /')        /(    )\.--.     /`-.
;  ( /        \  )  (    ._.'   / _  \
;   ))        ) (    `-.`.    ( '-' (
;   )'._.-.  \  )  ._ (  \   ) ._.'
;  (       )  ) \ (  '.)  ) (  '
;   )/.___/    )/  '._.__/   )/
;
;  (quote
;     (an (elegant (weapon
;         (for (a (more
;             (civilized age))))))))

;;;    ____ _  _ _  _ ____ ___ _ ____ _  _ ____
;;;   |___ |  | |\ | |     |  | |  | |\ | [__
;;;   |    |__| | \| |___  |  | |__| | \| ___]

(defmethod str2thing (x) x)
(defmethod str2thing ((x string))
  "coerce `x` from a string to a non-string"
  (let ((x (string-trim '(#\Space #\Tab) x)))
    (if (equal x "?")
      #\?
      (let ((y (ignore-errors (read-from-string x))))
        (if (numberp y) y x))))))

(defun str2list (s &optional (sep #\,) (x 0) (y (position sep s :start (1+ x))))
  "divide `s` on `sep`"
  (cons (subseq s x y) (and y (str2list s sep (1+ y)))))

;; random stuff

(labels ((park-miller (&aux (multiplier 16807.0d0) (modulus 2147483647.0d0))
                      (setf *seed* (mod (* multiplier *seed*) modulus))
                      (/ *seed* modulus)))
  (defun randf (&optional (n 1)) (* n (- 1.0d0 (park-miller))))
  (defun randi (&optional (n 1)) (floor (* n (park-miller)))))

(defun triangle (&optional (c .5) &aux (u (randf)) (v (randf)))
  "Return sample from triangular distribution doi.org/10.1016/j.mcm.2008.06.013"
  (+ (* (- 1 c) (min u v)) (* c (max u v))))

(defun normal (&optional (mu 0) (sd 1))
  "Return sample from normal distribution"
  (+ mu (* sd (sqrt (* -2 (log (randf)))) (cos (* 2 pi (randf))))))

;; stats

(defun per (seq &optional (p .5) &aux (v (coerce seq 'vector)))
  "Return `p`-th item from seq"
  (elt v (floor (* p (length v)))))

(defun sd (seq &optional (key #'identity))
  "Find sd from a sorted list"
  (/ (- (funcall key (per seq .9)) (funcall key (per seq .1))) 2.56))

(defun ent (alist &aux (n 0) (e 0))
  "Return entropy of symbols in an assoc list"
  (dolist (two alist) (incf n (cdr two)))
  (dolist (two alist e) (let ((p (/ (cdr two) n))) (decf e (* p (log p 2))))))

;; main command stuff

(defun ok (test msg)
  "handle tests within a test function"
  (cond (test (format t "~aPASS ~a~%" #\Tab  msg))
        (t    (incf *fails* )
              (if (? dump)
                (assert test nil msg)
                (format t "~aFAIL ~a~%" #\Tab msg)))))

(defun main (&aux (defaults (copy-tree *options*)))
  "Update *options* from command-line. Run the test suite."
  (labels ((stop () #+clisp (exit *fails*)
                    #+sbcl  (sb-ext:exit :code *fails*))
           (args () #+clisp ext:*args*
                    #+sbcl  sb-ext:*posix-argv*)
           (trim (x) (string-left-trim '(#\Space #\Tab) x))
           (show  (lst)
                  (terpri)
                  (dolist (line (str2list (cadr lst) #\Newline 0))
                    (format t "~&~a~%" (trim line)))
                  (loop for (slot (flag help b4)) on (cddr lst) by #'cddr do
                    (format t " ~a ~a = ~a~%" flag help b4)))
           (cli  (flag b4 &aux (x (member flag (args) :test #'equal)))
                 (cond ((not x) b4 )
                       ((eq b4 t) nil)
                       ((eq b4 nil) t)
                       (t (str2thing (elt x 1)))))
           (test (todo)  (print 1) (when (fboundp todo)
                                     (format t "~a~%" (type-of todo))
                                     (setf *seed* (? seed))
                                     (funcall todo)
                                     (setf *options* (copy-tree defaults)))))
    (loop for (slot (flag help b4)) on (cddr *options*) by #'cddr do
      (setf (getf *options* slot) (list flag help (cli flag b4))))
    (if (? help)
      (show *options*)
      (dolist (todo (if (equalp "all" (? todo)) *tests* (list (? todo))))
        (test (find-symbol (string-upcase todo)))))
    (stop)))
```

```lisp

;;;    ____ ___ ____ _  _ ____ ___ ____
;;;    [__   |  |__/ |  | |     |  [__
;;;    ___]  |  |  \ |__| |___  |  ___]

(defmethod ako ((s symbol) kind) (ako (symbol-name s) kind))
(defmethod ako ((s string) kind)
  "given a column header, comment on its the propertoes of that column"
  (let
    ((l1 '((ignore #\:) (klass #\!) (less #\-) (more #\+) (goal #\+ #\- #\!)))
     (l2 '((num #\$))))
    (and (> (length s) 2)
         (or (member (char s (1- (length s))) (cdr (assoc kind l1)))
             (member (char s 0)               (cdr (assoc kind l2)))))))
;;       ___ _  _ _ __
;;      (_-<| || | '  \
;;      /__/ \_, |_|_|_|
;;           |__/
(defstruct (sym  (:constructor %make-sym )) (n 0) at name all mode (most 0))

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

(defmethod add ((self sym) x)
  (with-slots (n all mode most) self
    (unless (eq x #\?)
      (incf n)
      (let ((now (incf (has x all))))
        (if (> now most)
          (setf most now
                mode x)))))
  x)

(defmethod div ((self sym)) (ent (sym-all self)))
(defmethod mid ((self sym)) (sym-mode self))
;;       _ _   _  _   _ __
;;      | ' \ | || | | '  \
;;      |_||_| \_,_| |_|_|_|

(defstruct (num  (:constructor %make-num )) (n 0) at name
  (all (make-array 5 :fill-pointer 0))
  (size (? enough))
  ok w (hi -1E32) (lo 1E32))

(defun make-num (&optional (at 0) (name ""))
  (%make-num :at at :name name :w (if (ako name 'less) -1 1)))

(defmethod add ((self num) x)
  (with-slots (n lo hi ok all size) self
    (unless (eq x #\?)
      (incf n)
      (setf lo (min x lo)
            hi (max x hi))
      (cond ((< (length all) size)  (vector-push-extend x all) (setf ok nil))
            ((< (randf) (/ size n)) (setf (elt all (randi (length all))) x
                                          ok nil)))))
  x)

(defmethod holds ((self num))
  (with-slots (ok all) self
    (unless ok (setf all (sort all #'<)))
    (setf ok t)
   all))

(defmethod div ((self num)) (sd  (holds self)))
(defmethod mid ((self num)) (per (holds self)))
;;                   _
;;       __   ___   | |  ___
;;      / _| / _ \  | | (_-<
;;      \__| \___/  |_| /__/

(defstruct (cols (:constructor %make-cols)) all x y klass)

(defun make-cols (names &aux (at -1) x y klass all)
  (dolist (name names (%make-cols :all (reverse all) :x x :y y :klass klass))
    (let* ((what (if (ako name 'num)  #'make-num #'make-sym))
           (now  (funcall what (incf at) name)))
      (push now all)
      (when (not (ako name 'ignore))
        (if (ako name 'goal)  (push  now x) (push now y))
        (if (ako name 'klass) (setf klass now))))))
;;       ___   __ _   ___
;;      / -_) / _` | (_-<
;;      \___| \__, | /__/
;;            |___/

(defstruct (egs  (:constructor %make-egs )) rows cols)

(defun make-egs (&optional from)
  (let ((self (%make-egs)))
    (cond ((consp from)
           (dolist (row from) (add self row)))
          ((stringp from)
           (print 22)
           (with-csv (row from)
             (print (make-cols (mapcar #'str2thing (str2list row))))
             (return-from make-egs nil))))
             ;(add self (mapcar #'str2thing (str2list row))))))
    self))

(defmethod add ((self egs) row)
  (with-slots (cols rows) self
    (if cols
      (push (mapcar #'add cols row) rows)
      (setf cols (make-cols row))))
  row)
;;;   _  _ _  _ _ ___    ___ ____ ____ ___ ____
;;;   |  | |\ | |  |      |  |___ [__   |  [__
;;;   |__| | \| |  |      |  |___ ___]  |  ___]

(deftest .cells () (print (mapcar #'str2thing (str2list "23,asda,34.1"))))

(deftest .has ()
  (let (x)
    (incf (has 'aa x))
    (incf (has 'aa x))
    (print x)
    (ok (eql 2 (cdr (assoc 'aa x))) "inc assoc list")))

(deftest .csv (&aux (n 0))
  (with-csv (row (? file)) (incf n))
  (ok (eq 399 n) "reading lines"))

(deftest .normal ()
  (dolist (n '(10000 5000 2500 1250 500 250 125 60 30 15))
    (let (l)
      (setf l (dotimes (i n (sort l #'<)) (push (normal) l)))
      (format t "~5@A : ~6,4f : ~6,4f ~%"  n (sd l) (per l)))))

(deftest .rand (&aux l)
  (dotimes (i 50) (push (randi 4) l))
  (print (sort l #'<)))

(deftest .ent ()
  (let (x)
    (incf (has 'this x) 4)
    (incf (has 'that x) 2)
    (incf (has 'other x) 1)
    (ok (<= 1.378 (ent x) 1.379) "diversity")))

(deftest .num (&aux (num (make-num)))
  (dotimes (i 100000 (print (holds num))) (add num i)))

(deftest .sym (&aux (sym (make-sym)))
  (dotimes (i 100000 (print (sym-all sym))) (add sym (randi 10))))

(deftest .cols (&aux c)
  (setf c (make-cols '("$ss" "age!" "$weight-")))
  (print c))

(deftest .egs ()
 (print 1000000)
 (make-egs (? file)))

;;;;------------------------------------------------------------------------
(main)

;              .---------.
;              |         |
;           -= _________ =-
;               ___   ___
;              |   )=(   |
;               ---   ---
;                  ###
;                #  =  #            "This ain't chemistry.
;                #######             This is art."

```