```lisp
; brknbad: explore the world better, explore the world for good.
; (c) 2022, Tim Menzies
;
;       .-------.
;       | Ba    | Bad <----.  planning= (better - bad)
;       |    56 |          |  monitor = (bad - better)
;       .-------.------.   |
;               | Be   |   v
;               |    4 | Better
;               .------.
(defun thing (x)
  (cond ((not (stringp x)) x)
        ((equal x "?")     #\?)
        (t (let ((y (ignore-errors (read-from-string x))))
             (if (numberp y) y x)))))

(defun cli (key flag help b4)
  "if the command line has ‘flag‘, update ‘b4‘."
  (let* ((args #+clisp ext:*args*
               #+sbcl  (cdr sb-ext:*posix-argv*))
         (it   (member flag args :test #'equal)))
    (list key flag help
          (if (not it)
            b4
            (if (eq b4 t) nil (if (eq b4 nil) t (thing (elt it 1))))))))

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

OPTIONS:")
  (cli 'cautious  "-c"  "abort on any error     "  t)
  (cli 'dump      "-d"  "stack dumps on error    "  nil)
  (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        "  "nothing")))

;      ._ _    _.   _  ._   _    _
;      | | |  (_|  (_  |   (_)  _>
; short hand for querying options
(defmacro !! (x)
  `(third (cdr (assoc ',x *options* :test #'equal))))

; print options
(defun show-options (o)
  (format t "~&~a~%" (second (car o)))
  (dolist (x (cdr o)) (format t " ~a ~a = ~a~%" (elt x 1) (elt x 2) (elt x 3))))

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

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

;      _  _|_  ._  o  ._    _      )    _|_  |_   o  ._    _
;     _>   |_  |   |  | |  (_|    /_     |_  | |  |  | |  (_|
;                           _|                              _|
; return string `s` divided on comma
(defun cells (s &optional (x 0) (y (position #\, s :start (1+ x))))
  (cons (string-trim '(#\Space #\Tab) (subseq s x y))
        (and y (cells s (1+ y)))))

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

;      ._   _.  ._    _|   _   ._ _
;      |   (_|  | |  (_|  (_)  | | |
(defvar *seed* (!! seed))
(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)))))

;      |  o   _  _|_      _.         _   ._
;      |  |  _>   |_     (_|  |_|  (/_  |    \/
;                          |                 /
(defun normal (&optional (mu 0) (sd 1))
  (+ mu (* sd (sqrt (* -2 (log (randf)))) (cos (* 2 pi (randf))))))

(defun per (seq &optional (p .5) &aux (v (coerce seq 'vector)))
  (elt v (floor (* p (length v)))))

(defun sd (seq &optional (key #'identity))
  (/ (- (funcall key (per seq .9)) (funcall key (per seq .1))) 2.56))

(defun ent (alist &aux (n 0) (e 0))
  (dolist (two alist) (incf n (cdr two)))
  (dolist (two alist e) (let ((p (/ (cdr two) n))) (decf e (* p (log p 2))))))
;                                                  _
;      |_    _    _.   _|   _   ._    o  ._   _|_   _
;      | |  (/_  (_|  (_|  (/_  |     |  | |   |   (_)
(defmethod ako ((s symbol) kind) (ako (symbol-name s) kind))
(defmethod ako ((s string) kind)
  (let
    ((l1 '((ignore #\:) (klass #\!) (less #\-) (more #\+) (goal #\+ #\- #\!)))
     (l2 '((num #\$))))
    (or (member (char s (1- (length s))) (cdr (assoc kind l1)))
        (member (char s 0)               (cdr (assoc kind l2))))))
;       _        ._ _
;      _>  \/   | | |
;          /
; check for certain `kind`s or suffixes or prefixes
(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 div ((self num)) (sd  (holds self)))
(defmethod mid ((self num)) (per (holds self)))

(defmethod holds ((self num))
  (with-slots (ok all) self
    (unless ok (setf all (sort all #'<)))
    (setf ok t)
    all))
;      _   _   |   _
;     (_  (_)  |  _>
(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)
           (with-csv (row (!! files)) (add self (mapcar #'thing (cells 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)
```

```lisp
;     _|_   _    _  _|_   _
;      |   (/   _>   |_  _>
(defvar *tests* nil)
(defvar *fails* 0)

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

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

(defun tests (&aux (defaults (copy-tree *options*)))
  (dolist (todo (if (equalp "all" (!! todo)) *tests* (list (!! todo))))
    (setf todo (find-symbol (string-upcase todo)))
    (when (fboundp todo)
      (format t "~a~%" todo)
      (setf *seed* (!! seed))
      (funcall todo)
      (setf *options* (copy-tree defaults))))
  #+clisp (exit *fails*)
  #+sbcl  (sb-ext:exit :code *fails*))

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

(deftest .has ()
  (let (x y)
    (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))

;      _     _  _|_   _   . _ _
;     _>  \/  _>   |_  (/_  | | |
;         /
(if (!! help) (show-options *options*) (tests))
```