```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
;                .------.

; TODO: move cols from push+reverse to mapcar

(defmethod thing (x)  x)
(defmethod thing ((x string))
  (if (equal x "?")
    x
    (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))))))))

(defparameter *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 'enough    "-e"  "enough items for a sample "  512)
  (cli 'far       "-F"  "far away                "  .9)
  (cli 'file      "-f"  "read data from file     "  "../data/auto93.lisp")
  (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         "  "")))

;      ._ _    _.   _   ._    _    _
;      | | |  (_|  (_   |    (_)  _>
; 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))))))

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

(defun %with-csv (file)
  (with-open-file (str file)
    (loop (cells  (or (read-line str nil) (return-from %csv))))))

;      _  _|_  ._  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)))))

;      ._   _.  ._    _|   _   ._ _
;      |   (_|  | |  (_|  (_)  | | |
(defvar *seed* (!! seed))
(labels ((park-miller (&aux (multipler 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 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 (elt two 1)))
  (dolist (two alist e) (let ((p (/ (elt two 1) n))) (decf e (* p (log p 2))))))
;                                                _
;      |_    _    _.   _|   _   ._    o  ._   _|_   _
;      | |  (/_  (_|  (_|  (/_  |     |  | |   |   (_)
(defun ako (x kind)
  (let
    ((l1 '((ignore #\:) (klass #\!) (less #\-) (more #\+) (goal #\+ #\- #\!)))
     (l2 '((num #\$)))
     (s  (symbol-name x)))
    (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-num :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)
;      ._        ._ _
;      | |  |_|  | | |
(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-sym :at at :name name :w (if (ako name 'less) -1 1)))

(defmethod add ((self num) x)
  (with-slots (n lo hi all size) self
    (unless (eq x #\?)
      (incf n)
      (setf lo (min x lo)
            hi (max x hi))
      (cond ((< (length all) size)  (vector-push x all) (setf ok nil))
            ((< (randf) (/ size n)) (setf (elt (randi (length all))) x
                                                   ok nil)))))
  x)
;       _    _   |   _
;      (_   (_)  |  _>
(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-name #'make-sym))
           (now  (funcall what (incf at) name)))
      (push now all)
      (when (not (ako name 'ignore))
        (if (ako name 'goal) (push x now) (push y now))
        (if (ako name 'klass) (setf klass now))))))

;       _    _    _
;      (/_  (_|  _>
;            _|
(defstruct (egs  (:constructor %make-egs )) rows cols)

(defun make-egs (&optional from)
  (let ((self (%make-egs)))
    (cond ((stringp from)
           (dolist (row (csv (!! files))) (add self row)))
          ((consp from)
           (dolist (row from) (add self 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)
  (format t " ~a ~a" (if test "PASS" "FAIL") msg)
  (unless test
    (incf *fails* )
    (if  (!! dump)) (assert test nil 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 (!! todo) (list (!! todo)) *tests*))
    (setf *seed* (!! seed))
    (funcall todo)
    (setf *options* (copy-tree defaults)))
  #+clisp (exit *fails*)
  #+sbcl  (sb-ext:exit :code *fails*))

;      _       _   _|_   _    ._ _
;     _>  \/  _>    |_  (/_   | | |
;         /
(defun make () (load 'bnb))
```