```lisp
; vim: ts=2 sw=2 et:
(defpackage :small (:use :cl))
(in-package :small)
(defstruct our
  (help
"sbcl --script lib.lisp [OPTIONS
(c) 2022, Tim Menzies, MIT license

Lets have some fun.")
  (options
    '((enough  "-e" "enough items for a sample"   512)
       (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        "  "")))
  (copyright "
Copyright (c) 2022 Tim Menzies
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS'
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
AUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."))

(defvar *config* (make-our))
```

```lisp
;     ___
;   /'___\
;  /\ \__/      __  __       ____       ____
;  \ \ ,__\    /\ \/\ \     /'  _ `\   /',__\
;   \ \ \_/    \ \ \_\ \   /\ \/\ \   /\__, `\
;    \ \_\      \ \____/   \ \_\ \_\  \/\____/
;     \/_/       \/___/     \/_/\/_/   \/___/

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

(defmacro aif (test yes &optional no)
  "Anaphoric if (traps result of conditional in 'it')."
  `(let ((it ,test)) (if it ,yes ,no)))

(defmacro whale (expr &body body)
  "Anaphoric while (traps result of conditional in 'a')."
  `(do ((a ,expr ,expr)) ((not a)) ,@body))

(defmacro ? (s x &rest xs)
  "Nested access to slots."
  (if (null xs) `(slot-value ,s ',x) `(? (slot-value ,s ',x) ,@xs)))

(defmacro $ (x &optional (our *config*))
  "Access  a config variable name."
  `(fourth (assoc ',x (our-options ,our))))

(defmacro with-csv ((lst file &optional out) &body body)
  "File row iterator."
  `(progn (csv ,file #'(lambda (,lst) ,@body)) ,out))

(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 randf (&optional (n 1.0))
  (setf ($ seed)  (mod (* 16807.0d0 ($ seed)) 2147483647.0d0))
  (* n (- 1.0d0 (/ ($ seed)                   2147483647.0d0))))

(defun randi (&optional (n 1)) (floor (* n (/ (randf 1000000.0) 1000000))))

(defun rnd (number &optional (places 3) &aux (div (expt 10 places)))
  (float (/ (round (* number div)) div)))

(defmethod rnds ((vec vector) &optional (places 3))
  (rnds (coerce  vec 'list) places))

(defmethod rnds ((lst cons) &optional (places 3))
  (mapcar #'(lambda (x) (rnd x places)) lst))
;  ____  ___  ____  _  _  _  ____  ____  ----------------------------------------------
; [__     |   |__/  |  |\ |  | __  [__
; ___]    |   |  \  |  | \|  |__]  ___]

(defun trim (x)
  "Remove whitespace front and back."
  (string-trim '(#\Space #\Newline #\Tab) x))

(defun num?(x)
  "Return a number, if you can. Else return trimmed string."
  (let ((y (ignore-errors (read-from-string x))))
    (if (numberp y) y (let ((x (trim x)))
                         (if (equal x "?") #\? x)))))

(defun subseqs (s &optional (sep #\,) (n 0))
  "Separate string on 'sep'."
  (aif (position sep s :start n)
    (cons (subseq s n it) (subseqs s sep (1+ it)))
    (list (subseq s n))))
;  ____   ____  ---------------------------------------------------------------------
; |    |  [__
; |____|  .___]  .

(defun args ()
  "Return list of command line arguments."
  #+clisp (cdddr (cddr (coerce (EXT:ARGV) 'list)))
  #+sbcl  (cdr sb-ext:*posix-argv*))

(defun csv (file &optional (fn #'print))
  "Send to 'fn' one list from each line."
  (with-open-file (str file)
    (loop (funcall fn (subseqs (or (read-line str nil) (return-from csv)))))))

(defun cli (&optional (our (make-our)) (lst (args)))
  "Maybe update 'our' with data from command line."
  (labels ((cli1 (flag x) (aif (member flag lst :test #'equalp)
                             (cond ((equal x t)   nil) ; flip boolean
                                   ((equal x nil) t)   ; flip boolean
                                   (t              (or (num? (second it)) x)))
                             x)))
    (dolist (x (our-options our) our)
      (setf (fourth x) (cli1 (second x) (fourth x))))))
;  ____  _  _  ____  ----------------------------------------------------------------
; |    | |  | |__/
; |____| |__| |  \

(defmethod print-object ((o our) s)
  (format s "~a~%~%OPTIONS:~%" (our-help o))
  (dolist (x (our-options o))
    (format s " ~5a ~a = ~a~%" (second x) (third x) (fourth x))))
```

```lisp
;           __                                      __
;          /\ \__                                  /\ \__
;    ____  \ \ ,_\    _ __   __  __     ___   \ \ ,_\     ____
;   /',__\  \ \ \/   /\`'__\/\ \/\ \   /'___\  \ \ \/    /',__\
;  /\__, `\  \ \ \_  \ \ \/ \ \ \_\ \ /\ \__/   \ \ \_  /\__, `\
;  \/\____/   \ \__\  \ \_\  \ \____/ \ \____\   \ \__\ \/\____/
;   \/___/     \/__/   \/_/   \/___/   \/____/    \/__/  \/___/
;

; ____ ____ _ _ _ -------------------------------------------------------------
; |___ |___ | | |
; |    |___ |_|_|

(defstruct (few (:constructor %make-few))
  ok (n 0) (lst (make-array 5 :adjustable t :fill-pointer 0)) (max ($ enough)))

(defun make-few (&key init) (adds (%make-few) init))

(defmethod add1 ((f few) x)
  (with-slots (max ok lst n) f
    (cond ((< (length lst) max)
           (setf ok nil)
           (vector-push-extend x lst))
          (t (if (< (randf)  (/ n max))
               (setf ok nil
                     (svref lst (floor (randi (length lst)) 1)) x))))))

(defmethod div ((f few)) (/ (- (per f .9) (per f .1)) 2.56))

(defmethod has ((f few))
  (with-slots (ok lst) f
    (unless ok (setf lst (sort lst #'<)
                     ok  t))
    lst))

(defmethod mid ((f few)) (per f .5))
(defmethod per ((f few) &optional (p .5) &aux (all (has f)))
  (aref (? f lst) (floor (* p (length (? f lst))))))
; _  _ _  _ _  _ --------------------------------------------------------------
; |\ | |  | |\/|
; | \| |__| |  |

(defstruct (num (:constructor %make-num))
  (n 0) (w 1) (at 0) (txt "") (all (make-few))
  (lo most-positive-fixnum) (hi most-negative-fixnum))

(defun make-num (&key init (txt "") (at 0) )
  (adds (%make-num :txt txt :at at :w (if (find #\< txt) -1 1)) init))

(defmethod add1 ((n num) x)
  (with-slots (n lo hi all) n
    (add all x)
    (incf n)
    (setf lo (min x lo)
          hi (max x hi))))

(defmethod div ((f num)) (div (? f all)))
(defmethod mid ((f num)) (mid (? f all)))
; ____ _   _ _  _ --------------------------------------------------------------
; [__   \_/  |\/|
; ___]   |   |  |

(defstruct (sym (:constructor %make-sym))
  mode seen (n 0) (at 0) (txt "") (most 0))

(defun make-sym (&key init (txt "") (at 0) )
  (adds (%make-sym :txt txt :at at) init))

(defmethod add1 ((s sym) x)
  (with-slots (n seen most mode) s
    (let ((now (inca x seen)))
      (if (> now most)
        (setf most now
              mode x)))))

(defmethod div ((f sym))
  (labels ((p    (x) (/ (cdr x) (? f n)))
           (plog (x) (* -1 (p x) (log (p x) 2))))
    (reduce '+ (mapcar #'plog (? f seen)))))

(defmethod mid ((f sym)) (? f mode))
; ____ _    _  _ ____ ---------------------------------------------------------
; | __ |    |  | |___
; |__] |___ |__| |___

(defun add (it x)
  (unless (eq x #\?)
    (incf (? it n))
    (add1 it x))
  x)

(defun adds (s lst) (dolist (new lst s) (add s new)))
;
;                               __
;    ___ ___       __         /\_\     ___
;  /' __` __`\   /'__`\       \/\ \  /' _ `\
;  /\ \/\ \/\ \  /\ \L\.\_      \ \ \ /\ \/\ \
;  \ \_\ \_\ \_\ \ \__/.\_\      \ \_\\ \_\ \_\
;   \/_/\/_/\/_/  \/__/\/_/       \/_/ \/_/\/_/

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

(defvar *tests* nil)
(defvar *fails* 0)

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

(defun demos (&optional what)
  (dolist (one *tests*)
    (let* ((what (string-upcase (string what)))
           (txt  (string-upcase (string one)))
           (doc  (documentation one 'function)))
      (when (or (not what) (search  what txt))
        (setf *config* (cli (make-our)))
        (multiple-value-bind (_ err)
          (ignore-errors (funcall one))
          (identity _)
          (incf *fails* (if err 1 0))
          (if err
            (format t "~&~a [~a] ~a ~a~%" "FAIL" one doc err)
            (format t "~&~a [~a] ~a~%"    "PASS" one doc)))))))

(dofun whale.(&aux (x '(1 2 3)))
  "whale"
  (whale (pop x) (print a)))

(dofun few.(&aux (f (make-few)))
  "few"
  (print (has (dotimes (i 10000 f) (add f (randi 100))))))

(dofun csv.(&aux head)
  "csv"
  (with-csv (line "../data/auto93.csv")
    (if head
      (format t "~s~%" (mapcar #'num? line))
      (setf head line))))

(dofun num.(&aux (n (make-num)))
  "streams of nums"
  (print (has (? (adds n '(1 2 4 #\? 1 1 1 1 1 11 )) all))))

(dofun sym.(&aux (s (make-sym)))
  "streams of symbols"
  (print (div (adds s (coerce "aaaabbc" 'list)))))
; ____ ___ ____ ____ ___ -----------------------------------------------------
; [__   |  |__| |__/  |
; ___]  |  |  | |  \  |
;
(setf *config* (cli (make-our)))
(if ($ help) (print *config*))
(if ($ license) (princ (our-copyright *config*)))
(demos ($ todo))
```