```
____ ___  ____ _  _ ___ _    _ ____ ___
|__| |__] |  | |  |  |  |    | [__  |__]
|  | |__] |__| |__|  | .|___ | ___] |

(defstruct+ about names all x y klass)

(defun make-about (lst)
  (let (all x y kl (pos -1))
    (dolist (s lst (%make-cols :names lst :all (reverse all) :x x :y y :klass kl))
      (let* ((what (if (eq #\$ (char s 0)) 'num 'sym))
             (col  (make-instance what s (incf pos))))
        (push col all)
        (unless (eq #\~ (charn s))
          (if (member (charn s) '(#\! #\- #\+)) (push  col y) (push  col x))
          (if (eq #\! (charn s)) (setf kl col)))))))


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

(defstruct+ data rows about)

(defun make-data (names &optional src (i (%make-data :about (make-about names))))
  (if (stringp src)
    (with-lines src (lambda (line) (add i (cells line))))
    (dolist (row src) (add i row)))
 i)

(defmethod clone ((d data) &optional src) (make-data (? d about names) src))


____ ____ _    _ ____ ___
|___ | __ |    | [__  |__]
|___ |__].|___ | ___] |

(load "tiny")
(in-package :tiny)

(print (make-row 12 '(1 2 3 4)))
(print (make-data '("$aa" "bb!~" "cc+")))
(print (! my 'seed))
(dotimes (i 20) (print (randi  200)))
; ; (defmethod clone ((d data) &optional src) (make-data (? d about names) src))
; ;(reads "../../data/auto93.lisp" 'print)
_    _ ___  _    _ ____ ___
|    | |__] |    | [__  |__]
|___ | |__].|___ | ___] |

;; hell
;;; macros
; ? obj x y z) == (slot-value (slot-value (slot-value obj 'x) 'y) 'z)
(defmacro ? (s x &rest xs)
 (if (null xs) `(slot-value ,s ',x) `(? (slot-value ,s ',x) ,@xs)))

;;; accessors
(defun ! (l x) (cdr (assoc x l)))

;;; string
; Last thing from a string
(defun charn (x) (char x (1- (length x))))

; Kill leading tailing whitespace.
(defun trim (x) (string-trim '(#\Space #\Tab #\Newline) x))

; Turn `x` into a number or string or "?"
(defmethod thing (x) x)
(defmethod thing ((x string))
  (let ((y (trim x)))
    (if (string= y "?") y
      (let ((z (ignore-errors (read-from-string y))))
        (if (numberp z) z y)))))

; Divide `str` on `char`, filtering all items through `filter`.
(defun splits (str &key (char #\,) (filter #'identity))
  (loop for start = 0 then (1+ finish)
    for        finish = (position char str :start start)
    collecting (funcall filter (trim (subseq str start finish)))
    until      (null finish)))

; String to lines or cells of things
(defun lines (string) (splits string :char   #\Newline))
(defun cells (string) (splits string :filter #'thing))

; Call `fun` for each line in `file`.
(defun with-lines (file fun)
  (with-open-file (s file)
    (loop (funcall fun (or (read-line s nil) (return))))))

;;; maths
; Random number control (since reseeding in LISP is... strange).
(defvar *seed* 10013)
(defun randi (&optional (n 1)) (floor (* n (/ (randf 1000000000.0) 1000000000))))
(defun randf (&optional (n 1.0))
  (setf *seed* (mod (* 16807.0d0 *seed*) 2147483647.0d0))
  (* n (- 1.0d0 (/ *seed* 2147483647.0d0))))

;;; settings
; Update `default` from command line (if it contains `flag` or `key`).
; CLI flags for booleans flip the setting (so they need no following arg).
(defun cli (key.flag.help.default)
  (destructuring-bind (key flag help default) key.flag.help.default
    (let* ((args #+clisp ext:*args*
                 #+sbcl sb-ext:*posix-argv*)
           (it (member flag args :test 'equalp)))
      (cons key (cond ((not it)              default)
                      ((equal default t)     nil)
                      ((equal default nil) t)
                      (t                     (thing (second it))))))))

; Update settings. If  `help` is set, print help.
(defun settings (header options)
  (let ((tmp (mapcar #'cli options)))
    (when (! tmp 'help)
     (format t "~&~%~{~a~%~}~%OPTIONS:~%" (lines header))
     (dolist (one options)
       (format t " ~a  ~a = ~a~%" (second one) (third one) (fourth one))))
    tmp))

;;;  defstruct+
; Creates %x for base constructor, enables pretty print, hides private slots
; (those starting with "_").
(defmacro defstruct+ (x &body body)
  (let* ((slots  (mapcar    (lambda (x) (if (consp x) (car x) x))          body))
         (public (remove-if (lambda (x) (eq #\_ (char (symbol-name x) 0))) slots)))
    `(progn
       (defstruct (,x (:constructor ,(intern (format nil "%MAKE-~a" x)))) ,@body)
       (defmethod print-object ((self ,x) str)
         (labels ((fun (y) (format nil ":~(~a~) ~a" y (slot-value self y))))
           (format str "~a" (cons ',x (mapcar #'fun ',public))))))))

;;; demos
; Define one demos.
(defvar *demos* nil)
(defmacro defdemo (what arg doc &rest src)
  `(push (list ',what ',doc (lambda ,arg ,@src)) *demos*))

; Run `one` (or `all`) the demos. Reset globals between each run.
; Return to the operating systems the failure count (so fails=0 means "success").
(defun demos (settings all &optional one)
  (let ((fails 0)
        (resets (copy-list settings)))
    (dolist (trio all)
      (let ((what (first trio)) (doc (second trio)) (fun (third trio)))
        (when (member what (list 'all one))
          (loop for (key . value) in resets do
            (setf (! settings key) value))
          (setf *seed* (or (! settings 'seed) 10019))
          (unless (eq t (funcall fun ))
            (incf fails)
            (format t "~&FAIL [~a] ~a ~%" what doc)))))
    #+clisp (exit fails)
    #+sbcl  (sb-ext:exit :code fails)))
_  _ _  _ _  _ _    _ ____ ___
|\ | |  | |\/| |    | [__  |__]
| \| |__| |  |.|___ | ___] |

(defstruct+ num  (txt "") (at 0) kept ok (w 1))
(defun make-num (s n) (%make-num :txt s :at n :w (if (equal #\- (charn s)) -1 1)))

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

(defstruct+ row  cells _about)

(defun make-row (about l)   (%make-row :cells l :_about about))


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

(defstruct+ sym  (txt "") (at 0) kept)

(defun make-sym (s n) (%make-sym :txt s :at n))

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

(defpackage :tiny (:use :cl) (:nicknames "tn"))
(in-package :tiny)
(load "lib")
(defvar my
  (settings "TOYIN: do stuff
     (c) 2022 Tim Menzies, BSD-2 clause license "
  '((file   "-f"   "help file                " "../../data/auto93.lisp")
    (help   "-h"   "show help                " nil)
    (keep   "-K"   "items to keep            " 256)
    (k      "-k"   "nb low attributes classes" 1)
    (m      "-m"   "nb low frequency classes " 2)
    (seed   "-s"   "random number seed       " 10019)
    (go     "-g"   "start up action          " ls))))

(mapcar #'load '("sym" "num" "row" "about" "data"))
___ _  _ ___  _    _ ____ ___
 |  |\/| |__] |    | [__  |__]
 |  |  | |   .|___ | ___] |

(defun a (lst)
  (destructuring-bind
    (b c d e)
    lst
    (format t "~%~%;;; => first:~a second:~a~&" b e)))

(a '(10 20 30 40))
```