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

;;;string
(defun chars (x) (if (symbolp x) (symbol-name x) x))
(defun char0 (x) (char (chars x) 0))
(defun charn (x) (let ((y (chars x))) (char y (1- (length y)))))

(defun trim (x) (string-trim '(#\Space #\Tab #\Newline) x))

(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)))))

(defun splits (string &key (sep #\,) (filter #'identity))
  (loop for start = 0 then (1+ finish)
    for       finish = (position sep string :start start)
    collecting (funcall filter (trim (subseq string start finish)))
    until      (null finish)))

(defun lines (string) (splits string :sep #\Newline))
(defun cells (string) (splits string :filter #'thing))

(defun with-lines (file fun)
  (with-open-file (s file)
    (loop (funcall fun (or (read-line s nil) (return))))))
;;; maths
(defvar *seed* 10013)
(defun randi (&optional (n 1)) (floor (* n (/ (randf 1e9) 1e9))))
(defun randf (&optional (n 1.0))
  (setf *seed* (mod (* 16807.0d0 *seed*) 2147483647.0d0))
  (* n (- 1.0d0 (/ *seed* 2147483647.0d0))))


;;; settings
(defun cli (lst)
  (destructuring-bind (key flag help default) lst
    (let* ((args #+clisp ext:*args* #+sbcl sb-ext:*posix-argv*)
           (it (or (member flag args :test 'equal)
                   (member key  args  :test 'equal))))
      (cons key (cond ((not it)              default)
                      ((equal default t)   nil)
                      ((equal default nil) t)
                      (t                     (thing (second it))))))))

(defun settings (header options)
  (let ((tmp (mapcar #'cli options)))
    (when (cdr (assoc 'help tmp))
     (format t "~&~%~[~a~%~]~%OPTIONS:~%" (lines header))
     (dolist (one options)
       (format t " ~a  ~a = ~a~%" (second one) (third one) (fourth one))))
    tmp))

;;;  defstruct+ : %x for base constructor, pretty print built in
(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)))))))

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

(defun demos (settings all-demos &optional want)
  (let ((fails 0)
        (resets (copy-list settings)))
    (dolist (trio all-demos)
      (let ((what (first trio)) (doc (second trio)) (fun (third trio)))
        (when (member what (list 'all want))
          (loop for (key . value) in resets do
            (setf (cdr (assoc key settings)) value))
          (setf *seed* (or (cdr (assoc 'seed settings)) 10019))
          (unless (eq t (funcall fun ))
            (incf fails)
            (format t "~&FAIL [~a] ~a ~%" what doc)))))
    #+clisp (exit fails)
    #+sbcl  (sb-ext:exit :code fails)))
```