```
; vim: ts=2 et sw=2 et:
;;;; macros ------------------------------------------------------------------
(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 $ (x &optional (our *config*))
  "Access  a config variable name."
  `(fourth (assoc ',x (our-options ,our))))

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

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

;;;; random  -----------------------------------------------------------------
(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))))

;;;; strings -----------------------------------------------------------------
(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))))

;;;; operating system --------------------------------------------------------
(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 (second (member flag lst :test #'equalp))
                               (cond ((equal x t)   nil) ; flip boolean
                                     ((equal x nil) t)   ; flip boolean
                                     (t             (or (num? it) x)))
                               x)))
    (dolist (x (our-options our) our)
      (setf (fourth x) (cli1 (second x) (fourth x))))))
```