

```

1 ;(defpackage :sublime (:use :cl))
2 ;(in-package :sublime)
3
4 ;; \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._
5 ;; ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
6 ;; \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._
7 ;; ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
8 ;; \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._ \_.._
9 ;; ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
10
11 ;; (quote
12 ;;   (an (elegant (weapon
13 ;;     (for (a (more
14 ;;       (civilized age)))))))
15 ;;
16 ;;
17 ;;
18 ;;
19 ;;
20 ;;
21
22 (defstruct cli key flag help value)
23 (defstruct options
24   (help
25    "sbcl --noinform --script expose.lisp [OPTIONS]
26    (c) 2022, Tim Menzies, MIT license
27
28    Lets have some fun.")
29   (list
30    (cli! 'cautious "-c" "about on any error" t)
31    (cli! 'enough "-c" "enough items for a sample" 512)
32    (cli! 'far "-f" "far away" .9)
33    (cli! 'file "-f" "read data from file" ".data/aut093.csv")
34    (cli! 'help "-h" "show help" nil)
35    (cli! 'license "-l" "show license" nil)
36    (cli! 'p "-p" "euclidean coefficient" 2)
37    (cli! 'seed "-s" "random number seed" 10019)
38    (cli! 'todo "-t" "start up action" "")))
39
40
41 (defmethod print-object ((c cli) s)
42   (with-slots (key flag help value) c
43     (format s "~5a ~a" flag help)
44     (if (member value '(t nil)) (terpri s) (format s "~4a~%" value))))
45
46 (defmethod print-object ((o options) s)
47   (with-slots (help options) o
48     (format s "~a~%~%OPTIONS~%" help)
49     (doalist (x options) (print-object (cdr x) s))))
50
51 (defun item (x)
52   "Return a number or a trimmed string."
53   (cond ((numberp x) x)
54         ((equal x "") nil)
55         (t (let ((y (ignore-errors (read-from-string x))))
56              (if (numberp y) y x)))))
57
58 (defun cli! (key flag help value)
59   (let* ((args (cdr sb-ext:*posix-argv*))
60          (it (member flag args :test #'equal)))
61     (if it (setf value (cond ((equal it t) nil)
62                              ((equal it nil) t)
63                              (t (item (second it)))))
64         (cons key (make-cli :key key :flag flag :help help :value value))))
65
66 (defvar *the* (make-options))
67 (defmacro $ (x) `(cli-value (cdr (assoc ',x (options-options *the*)))))

```

```

68 ;;
69 ;;
70 ;;
71 ;;
72 ;;
73 ;;
74 (defmacro aif (test y &optional n) `(let ((it ,test)) (if it ,y ,n)))
75 (defmacro ? (p x &rest xs) (if (null xs) `(getf ,p ',x) `(? (getf ,p ',x),@xs)))
76
77 ;; random
78 (defvar *seed* 10013)
79 (defun randi (&optional (n 1)) (floor (* n (/ (randf 1000.0) 1000))))
80 (defun randf (&optional (n 1.0))
81   (setf *seed* (mod (* 16807.0d0 *seed*) 2147483647.0d0))
82   (* n (- 1.0d0 (/ *seed* 2147483647.0d0))))
83
84 ;; lists
85 (defun nshuffle (lst)
86   "Return a new list that randomizes over of lst"
87   (let ((tmp (coerce lst 'vector)))
88     (loop for i from (length tmp) downto 2
89           do (rotatef (elt tmp (random i)) (elt tmp (1- i)))))
90   (coerce tmp 'list))
91
92 (defun per (lst &optional (p .5)) (elt lst (floor (* p (length lst)))))
93
94 ;; defthings
95 (defmacro defthing (x &rest slots &aux (id (gensym)))
96   "Defines structs with uniq ids 'id' and a constructor '(%make-x)'
97   and a print method that hides privates slots (those starting with '_')."
98   (labels ((hidep (z) (equal (char (symbol-name z) 0) #\_)))
99     (name (z) (if (consp z) (car z) z))
100     (names () (remove-if #'hidep (mapcar #'name slots)))
101     (%make () (intern (format nil "%MAKE--a" (symbol-name x)))))
102     `(let ((,id 0))
103       (defstruct (.x (:constructor ,(%make))) (_id (incf ,id)) ,@slots)
104       (defmethod print-object ((it ,x) s) (show-object it ',x ',(names) s))))
105
106 (defun show-object (it klass slots s)
107   (labels ((show (z) (let* ((k (intern (symbol-name z) "KEYWORD"))
108                             (v (slot-value it z)))
109                         (if v `',(k ,v) k))))
110     (print-object (cons klass (mapcar #'show slots)) s)))
111
112 ;; files
113 (defmacro with-csv ((lst file &optional out) &body body)
114   `(progn (%with-csv ,file (lambda (,lst) ,@body)) ,out))
115
116 (defun %csv (file &optional (fn 'print))
117   "Run a function 'fn' over file (sub-function of 'with-csv')."
118   (with-open-file (str file)
119     (loop (funcall fn (or (read-line str nil) (return-from %csv))))))

```

```

120 ;;
121 ;; l h a n e s
122 ;;
123 ;;
124 ;;
125
126 (defthing num (at 0) (txt "") (n 0) (w 1) (mu 0) (m2 0) (sd 0) max (ok t)
127   (lo most-positive-fixnum) (hi most-negative-fixnum)
128   (_has (make-array 32 :fill-pointer 0 :adjustable t)))
129
130 (defthing sym (at 0) (txt "") (n 0) has mode (most 0))
131 (defthing cols all x y klass)
132 (defthing sample rows cols)
133 (defthing range col lo hi has)
134
135 ;;
136 ;; l a u l l a
137 ;;
138
139 (labels ((lettern (x &aux (n (length x)) (and (> n 0) (subseq x (- n 1) n))))
140 (defun lessp (x) (equal "-" (lettern x)))
141 (defun morep (x) (equal "+" (lettern x)))
142 (defun klassp (x) (equal "!" (lettern x)))
143 (defun numo (x) (upper-case-p (char x 0)))
144 (defun goalp (x) (or (klassp x) (lessp x) (morep x))))
145
146 (defun make-num (n &optional (at 0) (txt ""))
147   (%make-num :at at :txt txt :max n :w (if (lessp txt) -1 1)))
148
149 (defun make-sym (&optional (at 0) (txt ""))
150   (%make-sym :at at :txt txt))
151
152 (defmethod add ((nu num) x)
153   (with-slots (lo hi max ok n _has) nu
154     (unless (null x)
155       (setf lo (min x lo)
156             hi (max x hi)
157             n (1+ n))
158       (cond ((> max (length _has))
159              (setf ok nil)
160              (vector-push-extend x _has))
161             ((< (randf) (/ max n))
162              (setf ok nil)
163              (elt _has (randi (length _has))) x))))
164   x)
165
166 (defmethod mid ((n num)) (per (has n) .5))
167 (defmethod div ((n num)) (/ (- (per (has n) .9) (per (has n) .1)) 2.56))
168
169 (defmethod has ((n num))
170   (with-slots (ok _has) n
171     (unless ok (setf ok t)
172       _has (sort _has #'<)))
173   _has))
174
175 ;;;; coe r c s
176 (defun str->items (s &optional (c #\,) (n 0) &aux (pos (position c s :start n)))
177   "Divide string 's' on character 'c'."
178   (if pos
179     (cons (item (subseq s n pos)) (str->items s (1+ pos)))
180     (list (item (subseq s n)))))

```

```

181 ;;
182 ;; l e s l s
183 ;;
184 ;;
185
186 (defvar *tests* nil)
187
188 (defmacro deftest (name params doc &body body)
189   `(progn (pushnew ',name *tests*) (defun ,name ,params ,doc ,@body)))
190
191 (defun demos (&optional what quit &aux (fails 0))
192   (dolist (one *tests* (if quit (exit :code fails)))
193     (let ((doc (documentation one 'function)))
194       (when (or (not what) (eql one what))
195         (setf *the* (make-options))
196         (setf *seed* ($ seed))
197         (multiple-value-bind
198           (_ err)
199             (if ($ cautious)
200                 (values (funcall one) nil)
201                 (ignore-errors (funcall one *the*)))
202           (identity _))
203         (incf fails (if err 1 0))
204         (if err
205             (format t "~&-&FAIL: [~a]~a~a~%" one doc err)
206             (format t "~&-&PASS: [~a]~a~a~%" one doc))))))
207
208
209 (defest aa? () "ads" (print 1))
210 (defest bb? () "ads" (print 2))
211
212 ;(defun file2sample (file &aux ((s (make-sample))))
213   ;;; lib
214   ;;; lists
215
216 (defun make () (load "sublime.lisp"))
217
218
219 ; file to samples
220 ; samples to clusters
221 ; clusters to ranges
222 ; ranges to tree

```