

```

1 ; vim: ts=2 sw=2 et:
2 ;
3 ;
4 ;
5 ;
6 ;
7 ;
8
9 (defpackage :small (:use :cl))
10 (in-package :small)

```

```

11 (defstruct our
12   (help
13     "sbel --noinform --script small.lisp [OPTIONS]
14     (c) 2022, Tim Menzies, MIT license
15
16     Lets have some fun.")
17   (options
18     '( (enough "-c" "enough items for a sample" 512)
19       (file "-f" "read data from file" ".../data/auto93.csv")
20       (help "-h" "show help" nil)
21       (license "-l" "show license" nil)
22       (p "-p" "euclidean coefficient" 2)
23       (seed "-s" "random number seed" 10019)
24       (todo "-t" "start up action" ""))
25     (copyright "
26     Copyright (c) 2022 Tim Menzies
27     All rights reserved.
28
29     Redistribution and use in source and binary forms, with or without
30     modification, are permitted provided that the following conditions are met:
31
32     1. Redistributions of source code must retain the above copyright notice, this
33     list of conditions and the following disclaimer.
34
35     2. Redistributions in binary form must reproduce the above copyright notice,
36     this list of conditions and the following disclaimer in the documentation
37     and/or other materials provided with the distribution.
38
39     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS'
40     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
41     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
42     DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
43     FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
44     DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
45     SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
46     CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
47     OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
48     OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."))
49
50 (defvar *config* (make-our))
51
52 ;
53 ;
54 ;
55 ;
56 ;
57
58 (defmacro aif (test yes &optional no)
59   "Anaphoric if (traps result of conditional in 'it')."
60   `(let ((it ,test)) (if it ,yes ,no)))
61
62 (defmacro while (expr &body body)
63   "Anaphoric while (traps result of conditional in 'a')."
64   `(do ((a ,expr ,expr)) ((not a)) ,@body))
65
66 (defmacro ? (s x &rest xs)
67   "Nested access to slots."
68   (if (null xs) `(slot-value ,s ',x) `(? (slot-value ,s ',x) ,@xs)))
69
70 (defmacro $ (x &optional (our *config*))
71   "Access a config variable name."
72   `(fourth (assoc ',x (our-options ,our))))
73
74 (defmacro with-csv ((lst file &optional out) &body body)
75   "File row iterator."
76   `(progn (csv ,file #'(lambda (,lst) ,@body)) ,out))
77
78 (defmacro inca (x a &optional (inc 1))
79   `(incf (cdr (or (assoc ,x ,a :test #'equal)
80     (car (setf ,a (cons (cons ,x 0) ,a)))))) ,inc))
81
82 (defmacro dofun (name params doc &body body)
83   `(progn (pushnew ',name *tests*)
84     (defun ,name ,params ,doc (progn (print ',name) ,@body))))

```

```

86 ;
87 ;
88 ;
89 ;
90 ;
91 ;
92 ;
93 ;
94 ;
95 ;
96 ;
97 ;
98 (defun randf (&optional (n 1.0))
99   (setf ($ seed) (mod (* 16807.0d0 ($ seed)) 2147483647.0d0))
100   (* n (- 1.0d0 (/ ($ seed) 2147483647.0d0)))
101 )
102 ;
103 (defun randi (&optional (n 1)) (floor (* n (/ (randf 1000000.0) 1000000))))
104 ;
105 (defun rnd (number &optional (places 3) &aux (div (expt 10 places)))
106   (float (/ (round (* number div)) div)))
107 ;
108 (defmethod rnds ((vec vector) &optional (places 3))
109   (rnds (coerce vec 'list) places))
110 ;
111 (defmethod rnds ((lst cons) &optional (places 3))
112   (mapcar #'(lambda (x) (rnd x places)) lst))
113 ;
114 ;
115 ;
116 ;
117 ;
118 ;
119 (defun trim (x)
120   "Remove whitespace front and back."
121   (string-trim '("#\Space #\Newline #\Tab") x))
122 ;
123 (defun num! (x)
124   "Return a number, if you can. Else return trimmed string."
125   (let ((y (ignore-errors (read-from-string x))))
126     (if (numberp y) y (let ((x (trim x)))
127       (if (equal x "") #? x))))
128 )
129 ;
130 (defun subseqs (s &optional (sep #\,) (n 0))
131   "Separate string on 'sep'."
132   (aif (position sep s :start n)
133     (cons (subseq s n it) (subseqs s sep (1+ it))))
134   (list (subseq s n)))
135 ;
136 ;
137 ;
138 ;
139 ;
140 ;
141 ;
142 ;
143 ;
144 ;
145 ;
146 ;
147 ;
148 ;
149 ;
150 ;
151 ;
152 ;
153 ;
154 ;
155 ;
156 ;
157 ;
158 ;
159 ;
160 ;
161 ;
162 ;
163 ;
164 ;
165 ;
166 ;

```

MATHS

```

98 (defun randf (&optional (n 1.0))
99   (setf ($ seed) (mod (* 16807.0d0 ($ seed)) 2147483647.0d0))
100   (* n (- 1.0d0 (/ ($ seed) 2147483647.0d0))))
101
102 (defun randi (&optional (n 1)) (floor (* n (/ (randf 1000000.0) 1000000))))
103
104 (defun rnd (number &optional (places 3) &aux (div (expt 10 places))
105   (float (/ (round (* number div)) div)))
106
107 (defmethod rnds ((vec vector) &optional (places 3))
108   (rnds (coerce vec 'list) places))
109
110 (defmethod rnds ((lst cons) &optional (places 3))
111   (mapcar #'(lambda (x) (rnd x places)) lst))
112
113 ;-----
114 ;STRINGS
115
116 (defun trim (x)
117   "Remove whitespace front and back."
118   (string-trim '(#\Space #\Newline #\Tab) x))
119
120 (defun num! (x)
121   "Return a number, if you can. Else return trimmed string."
122   (let ((y (ignore-errors (read-from-string x))))
123     (if (numberp y) y (let (x (trim x))
124       (if (equal x "?") #\? x))))))
125
126 (defun subseqs (s &optional (sep #\,) (n 0))
127   "Separate string on 'sep'."
128   (aif (position sep s :start n)
129     (cons (subseq s n it) (subseqs s sep (1+ it)))
130     (list (subseq s n))))

```

```
defmethod rnds ((lst cons) &optional (places 3))
  (mapcar #'(lambda (x) (rnd x places)) lst))
```

# STRINGS

```

116 (defun trim (x)
117   "Remove whitespace front and back."
118   (string-trim '("#\Space #\Newline #\Tab") x))
119
120 (defun num! (x)
121   "Return a number, if you can. Else return trimmed string."
122   (let ((y (ignore-errors (read-from-string x))))
123     (if (numberp y) y (let ((x (trim x)))
124                          (if (equal x "NaN" #\? x))))))
125
126 (defun subseqs (s &optional (sep #\,) (n 0))
127   "Separate string on 'sep'."
128   (if (position sep s :start n)
129       (cons (subseq s n it) (subseqs s sep (+ it)))
130       (list (subseq s n))))

```

```

131 ;
132 ;
133 ;
134 (defun args ()
135   "Return list of command line arguments."
136   #+clisp (cdddr (cdr (coerce (EXT:ARGV) 'list)))
137   #+sbcl (cdr sb-ext:*posix-argv*))
138 )
139 (defun csv (file &optional (fn #'print))
140   "Send to fn one list from each line."
141   (with-open-file (str file)
142     (loop (funcall fn (subseqs (or (read-line str) nil) (return-from csv))))))
143 )
144 (defun cli (&optional (our (make-our)) (lst (args)))
145   "Maybe update 'our' with data from command line."
146   (labels ((clil (flag x) (if (member flag lst :test #'equalp)
147                                (cond ((equal x t) nil) ; flip boolean
148                                      ((equal x nil) t) ; flip boolean
149                                      (t (or (num? (second it)) x))))
150            (dolist (x (our-options our) our)
151              (setf (fourth x) (clil (second x) (fourth x))))))
152 )

```

```
153 ; -----  
154 ; MISE  
155 ;  
156 (let ((_id 0))  
157       (defun id () (incf _id)))
```

158  
159 ;  
160 ;  
161 ;

```

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

```

167 ;  
168 ;  
169 ;  
170 ;  
171 ;  
172 ;  
173 ;

```

178 (defstruct (few
179   (constructor %make-few))
180   (ok n 0) (lst (make-array 5 :adjustable t :fill-pointer 0)) (max ($ enough)))
181
182 (defun make-few (&key init) adds init (%make-few))
183
184 (defmethod add1 ((f few) x)
185   (with-slots (max ok lst n) f
186     (cond ((< (length lst) max)
187            (setf ok nil)
188            (vector-push-extend x lst))
189           (t (if (< (randf) (/ n max))
190                  (setf ok nil
191                        (svref lst (floor (randi (length lst)) 1)) x))))))
192
193 (defmethod div ((f few) (/ (- (per f .9) (per f .1)) 2.56))
194
195 (defmethod has ((f few)
196   (with-slots (ok lst) f
197     (unless ok (setf lst (sort lst #'<)
198                          ok t))
199     lst))
200
201 (defmethod mid ((f few) (per f .5))
202 (defmethod per ((f few) &optional (p .5) &aux (all (has f)))
203   (aref (? f lst) (floor (* p (length (? f lst))))))

```

204 ;  
205 ;  
206 ;

```

207 (defstruct (num (:constructor %make-num))
208   (n 0) (w 1) (at 0) (txt "") (all (make-few))
209   (lo most-positive-fixnum) (hi most-negative-fixnum))
210
211 (defun make-num (skey init (txt "") (at 0)
212   (adds init (%make-num :txt txt :at :w (if (find #\< txt) -1 1))))

```

```

215 (defmethod addl ((n num) x)
216   (with-slots (n lo hi all) n
217     (add all x)
218     (incf n)
219     (setf lo (min x lo)
220             hi (max x hi))))
221
222 (defmethod div ((n num) (div (? n all)))
223 (defmethod mid ((n num) (mid (? n all)))
224 (defmethod norm ((n num) x)
225   (with-slots (lo hi) n
226     (if (< (- hi lo) 1e-32)
227       0
228       (/ (- x lo) (- hi lo))))
229
230 (defmethod dist2 ((n num) a b)
231   (cond ((and (eq a #?) (eq b #?)) 1)
232         ((eq a #?) (setf b (norm n a)
233                             a (if (> b 0.5) 1 0))
234         ((eq b #?) (setf a (norm n a)
235                             b (if (> a 0.5) 1 0))
236         (t (setf a (norm n a)
237                    b (norm n b))))
238   (abs (- a b)))

```

239 ;  
240 ;  
241 ;

```
242 (defstruct (sym (:constructor %make-sym))
243   mode seen (n 0) (at 0) (txt "") (most 0))
244
245 (defun make-sym (&key init (txt "") (at 0)
246   (adds init (%make-sym :txt txt :at at)))
```

```

248 (defmethod add1 ((s sym) x)
249   (with-slots (n seen most mode) s
250     (let ((now (incn x seen)))
251       (if (> now most)
252         (setf most now
253               mode x))))))
254
255 (defmethod dist2 ((c sym) x y) (if (eql x y) 0 1))
256
257 (defmethod div ((f sym))
258   (labels ((p (x) (/ (cdr x) (? f n)))
259            (plog x) (* -1 (p x) (log (p x) 2))))
260     (reduce '+ (mapcar #'plog (? f log))))))
261

```

```
262  
263 (defmethod mid ((f sym)) (? f mode))
```

264 ;  
265 ;  
266 :

```

267 (defmethod add ((it t) x)
268   (unless (eq x #\?)
269     (incf (? it n)
270       (add1 it x))
271     x)
272
273
274 (defmethod adds (lst s)
275   (dolist (new lst s) (add s new)))
276
277 (defun dist1 (col x y)
278   (if (and (eq x #\?) (eq y #\?))
279       1
280       (dist2 x y)))

```

```

281
282
283
284
285
286
287
288
289 ;
290 ; EXAMPLE
291 ;
292
293 (defstruct example cells)
294
295 (defmethod cell ((i example) (c integer)) (aref (? i cells) c))
296 (defmethod cell ((i example) c) (aref (? i cells) (? c at)))
297
298 (defmethod cells ((i example)) (? i cells))
299 (defmethod cells ((i cons)) i)
300
301 (defmethod it ((i example) (j example) cols)
302 (let ((s1 0) (s2 0) (n (length cols)))
303 (dolist (col cols (< (/ s1 n) (/ s2 n)))
304 (let ((a (norm col (cell i col)))
305 (b (norm col (cell j col))))
306 (decf s1 (exp (* (? col w) (/ (- a b) n))))
307 (decf s2 (exp (* (? col w) (/ (- b a) n)))))))
308
309 (defmethod dist ((i example) (j example) cols)
310 (let ((d 0) (n (length cols)))
311 (dolist (col cols (expt (/ d n) (/ 1 ($ p))))
312 (incf d (dist1 col (cell i col) (cell j col)))))
313 ;
314 ; SAMPLE
315 ;
316 (defstruct sample x y rows cols)
317
318 (defun skip? (s) (search "." s))
319 (defun goal? (s) (and (search "<" s) (search ">" s)))
320 (defun num? (s &aux (x (subseq s 0 1)))
321 (and (string<= "A" x) (string<= x "Z")))
322
323 (defmethod add ((s sample) eg)
324 (let ((n -1))
325 (with-slots (x y rows cols) s
326 (if cols
327 (push (mapcar #'add cols (cells eg)) rows)
328 (setf cols (mapcar #'(lambda (str) (col s (incf n) str)) (cells eg)))
329 eg)))
330
331 (defmethod col ((s sample) at str)
332 (let* ((what (if (num? str) #'make-num #'make-sym))
333 (now (funcall what :txt txt :at at)))
334 (unless (skip? str) (if (goal? str)
335 (push now (? s y))
336 (push now (? s x)))
337 now))
338 ;

```

```

338
339 ;
340 ;
341 ;
342 ;
343 ;
344
345 ;
346 ; DEMOS
347 ;
348
349 (defvar *fails* 0)
350 (defvar *tests* nil)
351 (defun demos (&optional what)
352 (dolist (one *tests*)
353 (let* ((what (string-upcase (string what)))
354 (txt (string-upcase (string one)))
355 (doc (documentation one 'function)))
356 (when (or (not what) (search what txt))
357 (setf *config* (cli (make-our)))
358 (multiple-value-bind (_ err)
359 (ignore-errors (funcall one)
360 (identity _))
361 (incf *fails* (if err 1 0))
362 (if err
363 (format t "~&-a[-a]~a~a-%" "FAIL" one doc err)
364 (format t "~&-a[-a]~a~a-%" "PASS" one doc))))))
365
366 (dofun whale.(&aux (x '(1 2 3)))
367 "whale"
368 (whale (pop x) (print a)))
369
370 (dofun few.(&aux (f (make-few)))
371 "few"
372 (print (has (dotimes (i 10000 f) (add f (randi 100))))))
373
374 (dofun csv.(&aux head (n 0))
375 "csv"
376 (with-csv (line "./data/auto93.csv")
377 (if (> (incf n) 5) (return-from csv. t))
378 (if head
379 (format t "~s~%" (mapcar #'num! line))
380 (setf head line))))
381
382 (dofun num.(&aux (n (make-num)))
383 "streams of nums"
384 (print (has (? (adds '(1 2 4 #\? 1 1 1 1 1 1) n) all))))
385
386 (dofun sym.(&aux (s (make-sym)))
387 "streams of symbols"
388 (print (div (adds (coerce "aaaabbc" 'list) s))))
389 ;
390 ; START
391 ;
392 ;
393 (setf *config* (cli (make-our)))
394 (if ($ help) (print *config*))
395 (if ($ license) (princ (our-copyright *config*)))
396 (demos ($ todo))
397

```