

```

1 (defpackage :run (:use :cl))
2 (in-package :run)
3
4 ;; ## Vars
5 (defvar *help* "
6 run: simple lisp
7 (c) 2023 Tim Menzies <tim@ieee.org> BSD-2
8 USAGE: lisp run-lisp [OPTIONS]
9
10 OPTIONS:
11 -h help show help      =nil
12 -g action start up action =none
13 -p p distance coefficient =2
14 -s seed random number seed =10013*
15
16 (defvar *save* nil)
17 (defvar *settings* nil)
18
19 ;; ## Lib
20 ;; ### macros / / / /- /- /- /- /-
21 (defmacro q (s)
22 "convenience function to access access settings"
23 (getf (car (member ',s *settings*) :key (lambda (x) (getf x :key))) :test #'equal)
24 :value))
25
26 (defmacro geta (x lst &optional (init 0))
27 "ensure that 'lst' includes a cell (x num) and return that cell"
28 '(cdr (or (assoc ,x ,lst :test #'equal)
29 (car (setf ,lst (cons (cons ,x ,init) ,lst))))))
30
31 ;; ## strings
32 (defun charn (s &optional (n 0))
33 "is 's' a string holding 'c' at position 'n'?"
34 (if (stringp s)
35 (if (< n 0)
36 (charn s (+ (length s) n))
37 (and (>= n 0) (< n (length s)) (eql c (char s n)))))
38
39 (defun trim (s)
40 "kill leading, trailing whitespace"
41 (string-trim '(:#\Space #\Tab #\Newline) s))
42
43 ;; ## things
44 (defun thing (s &aux (sl (trim s)))
45 "coerce 's' into a number or string or nil or nil or #?"
46 (cond ((equal sl "") #?)
47 ((equal sl "") t)
48 ((equal sl "nil") nil)
49 (t (let ((n (read-from-string sl nil nil)))
50 (if (numberp n) n sl))))
51
52 (defun subseqs (s &optional (sep #\,)) (filter #'thing (here 0))
53 "find subsequences from 's', divided by 'sep', filtered through 'filter'"
54 (let* ((there (position sep s :start here))
55 (word (funcall filter (subseq s here there))))
56 (labels ((tail () (if there (subseqs s sep filter (1+ there))
57 (if (equal word "") (tail) (cons word (tail))))))
58 (defun with-lines (file fun &optional (filter #'subseqs))
59 "Call 'fun' for each line in 'file'"
60 (with-open-file (s file)
61 (loop (funcall fun (funcall filter (or (read-line s nil) (return)))))))
62
63 ;; ## random
64 ; Unlike Common Lisp, these randoms let reset the seed.
65 (defvar *seed* 10013)
66 (defun rand (&optional (n 2))
67 "Random float 0.<n"
68 (setf *seed* (mod (* 16807.0d0 *seed*) 2147483647.0d0))
69 (* n (- 1.0d0 (/ *seed* 2147483647.0d0))))
70
71 (defun rint (&optional (n 2) &aux (base 1000000000.0d0))
72 "Random int 0..n"
73 (floor (* n (/ (rand base) base))))
74
75 ;; ## settings
76 (defun settings (s)
77 "for lines like --Key Flag ---Default--, return (KEY flag (thing Default))"
78 (loop :for (flag key . lst)
79 :in (subseqs s #\Newline (lambda (sl) (subseqs sl #\Space #\trim)))
80 :if (charn flag #\-)
81 :collect (list :key (intern(string-upcase key))
82 :value (thing(car (last lst))) (flag flag)))
83
84 (defun cli (settings &optional (args #+clisp ext:'args)
85 #+abcl ab-ext:'posix-argv*))
86 "update settings from command-line (non-boolean settings need a value after the flag)"
87 (boolean settings just expect a flag (and, if used on command line, this flips the default))
88 (dolist (setting settings)
89 (let ((b4 (getf setting :value))
90 (if now
91 (setf (getf setting :value) (cond ((eq b4 t) nil)
92 ((eq b4 nil) t)
93 (t (thing now)))))))
94
95 ;; ## eg
96 (defmacro eg (what fun)
97 "define an example"
98 '(push (list :name ',what :fun ,fun) *egs*))
99
100 (defun egs ()
101 "run 'all' actions or just the (! action) action
102 (recreating random seed and other settings before each action)"
103 (let ((fails 0)
104 (b4 (copy-list *settings*)))
105 (dolist (eg (reverse *egs*))
106 (let ((name (getf eg :name)))
107 (when (or (equal (! action) name)
108 (equal (! action) "all*"))
109 (setf *settings* b4
110 *seed* (! seed)
111 (format t "TESTING ~a" name)
112 (cond ((funcall (getf eg :fun)) (format t "PASS ~M~^M~^E-%")
113 (format t "FAIL ~M~^M~^L~^L-%")
114 (incf fails))))))
115 #+clisp (ext:exit fails)
116 #+abcl (ab-ext:exit :code fails)))
117
118 ;; ## egs and help
119 (defun about ()
120 "show the help string (built from 'help' and the doc strings from *egs*"
121 (format t "~-~%-~%ACTIONS~%-~%~% *help*")
122 (dolist (eg (reverse *egs*))
123 (format t " ~-g~%-~a~%-~%" (getf eg :name) (documentation (getf eg :fun) 'function))))
124
125 ;; ## Data
126 (defun isNum (s) (s) (and (> (length s) 1) (upper-case-p (char s 0))))
127 (defun isGoal (s) (or (isKlass s) (isLess s) (isMore s)))
128 (defun isTmore (s) (charn s #\X -1))
129 (defun isKlass (s) (charn s #\V -1))
130 (defun isLess (s) (charn s #\< -1))
131 (defun isMore (s) (charn s #\> -1))
132
133 ;; ## sym
134 (defstruct sym (at 0) (txt "")
135 "summarizes streams of numbers"
136 (make-sym :at :txt txt :w (if (isLess txt) -1 1)))
137
138 (defmethod add ((i sym) x)
139 (with-slots (n has mode most) i
140 (unless (eq x #?)
141 (incf n)
142 (incf (geta x has)
143 (when (> (geta x has) most)
144 (setf most (geta x has)
145 mode x))))))
146
147 (defmethod nif ((i sym) (sym-mode i))
148 (defmethod div ((i sym))
149 "Diversity (entropy)."
150 (with-slots (has n) i (labels ((fun (p) (* -1 (* p (log p 2))))
151 (loop for (_, n) in has sum (fun (/ n1 n))))))
152
153 (defmethod dist ((i sym) x y)
154 (cond ((and (equal x #?) (equal x #?)) 1)
155 (t (if (equal x y) 0 1)))
156
157 ;; ## num
158 (defstruct num (at 0) (txt "") (n 0) (mu 0) (m2 0) (w 1) (lo 1E31) (hi -1321))
159 (defun num (&optional (at 0) (txt ""))
160 (make-num :at :txt txt :w (if (isLess txt) -1 1)))
161
162 (defmethod add ((i num) x) ;; Add one thing, updating 'lo,hi'
163 (with-slots (n lo hi mu m2) i
164 (unless (eq x #?)
165 (let ((d (- x mu)))
166 (incf mu (/ d n))
167 (incf m2 (* d (- x mu)))
168 (setf lo (min x lo)
169 hi (max x hi))))))
170
171 (defmethod mid ((i num) (num-mu i))
172 (defmethod div ((i num))
173 (with-slots (n m2) i (if (<= n 1) 0 (sqrt (/ m2 (- n 1)))))
174
175 (defmethod norm ((i num) x) ;; Map 'x' 0..1 (unless unknown, unless too small)
176 (with-slots (lo hi) i
177 (if (eq x #?) x (/ (- x lo) (- hi lo 1e-32))))
178
179 (defmethod dist ((i num) x y)
180 (if (and (equal x #?) (equal x #?))
181 1
182 (let ((x (norm i x))
183 (y (norm i y)))
184 (if (eq x #?) (setf x (if (< y .5) 1 0)))
185 (if (eq y #?) (setf y (if (< x .5) 1 0)))
186 (abs (- x y)))))
187
188 (defstruct row cells y-used)
189 "create something that holds 'cells'"
190 (make-row :cells cells)
191
192 (defmethod th ((r row) (c num)) (elt (row-cells r) (num-at c))
193 (defmethod th ((r row) (c sym)) (elt (row-cells r) (sym-at c))
194 (defmethod th ((r row) (n number)) (elt (row-cells r) n))
195
196 ;; ## Cols
197 (defstruct cols all x y klass)
198 (defun cols (lst &aux (l (make-cols)) (at -1))
199 "factory for generating column headers from list of column names"
200 (with-slots (all x y klass) l
201 (dolist (txt lst)
202 (push col all)
203 (when (not (isGoal txt))
204 (if (isGoal txt) (push col y) (push col x)
205 (if (isKlass txt) (setf klass col))))))
206
207 (defmethod add ((i cols) row)
208 "update x and y column headers from data in row. returns row"
209 (dolist (col (cols-x i)) (add col (th row col)))
210 (dolist (col (cols-y i) row) (add col (th row col)))
211
212 ;; ## Data
213 (defstruct data rows cols)
214 (defun data (src &aux (l (make-data)))
215 "create data from either a file called 'src' or a list 'src'"
216 (labels ((update (x) (add i x))
217 (if (stringp src) (with-lines src #'update) (mapc #'update src)
218 i))
219 (defmethod add ((i data) x)
220 "make 'cols' (if currently missing) or update the cols and rows"
221 (with-slots (cols rows) i
222 (if cols
223 (push (add cols (if (row-p x) x (row! x)))
224 rows)
225 (setf cols (cols! x)))))
226
227 (defmethod dist ((i data) (row1 row) (row2 row))
228 "Returns 0..1"
229 (let ((d 0) (n 1E-32))
230 (dolist (col (cols-x (data-cols i))
231 (expt (/ d n) (/ 1 (! p))))
232 (incf d (expt (dist col row1 row2) (! p)))
233 (incf n)))
234
235
236
237
238
239
240
241
242
243
244

```

```

138 (defun sym (&optional (at 0) (txt ""))
139 "summarizes streams of numbers"
140 (make-sym :at :txt txt :w (if (isLess txt) -1 1)))
141
142 (defmethod add ((i sym) x)
143 (with-slots (n has mode most) i
144 (unless (eq x #?)
145 (incf n)
146 (incf (geta x has)
147 (when (> (geta x has) most)
148 (setf most (geta x has)
149 mode x))))))
150
151 (defmethod nif ((i sym) (sym-mode i))
152 (defmethod div ((i sym))
153 "Diversity (entropy)."
154 (with-slots (has n) i (labels ((fun (p) (* -1 (* p (log p 2))))
155 (loop for (_, n) in has sum (fun (/ n1 n))))))
156
157 (defmethod dist ((i sym) x y)
158 (cond ((and (equal x #?) (equal x #?)) 1)
159 (t (if (equal x y) 0 1)))
160
161 ;; ## num
162 (defstruct num (at 0) (txt "") (n 0) (mu 0) (m2 0) (w 1) (lo 1E31) (hi -1321))
163 (defun num (&optional (at 0) (txt ""))
164 (make-num :at :txt txt :w (if (isLess txt) -1 1)))
165
166 (defmethod add ((i num) x) ;; Add one thing, updating 'lo,hi'
167 (with-slots (n lo hi mu m2) i
168 (unless (eq x #?)
169 (let ((d (- x mu)))
170 (incf mu (/ d n))
171 (incf m2 (* d (- x mu)))
172 (setf lo (min x lo)
173 hi (max x hi))))))
174
175 (defmethod mid ((i num) (num-mu i))
176 (defmethod div ((i num))
177 (with-slots (n m2) i (if (<= n 1) 0 (sqrt (/ m2 (- n 1)))))
178
179 (defmethod norm ((i num) x) ;; Map 'x' 0..1 (unless unknown, unless too small)
180 (with-slots (lo hi) i
181 (if (eq x #?) x (/ (- x lo) (- hi lo 1e-32))))
182
183 (defmethod dist ((i num) x y)
184 (if (and (equal x #?) (equal x #?))
185 1
186 (let ((x (norm i x))
187 (y (norm i y)))
188 (if (eq x #?) (setf x (if (< y .5) 1 0)))
189 (if (eq y #?) (setf y (if (< x .5) 1 0)))
190 (abs (- x y)))))
191
192 (defstruct row cells y-used)
193 "create something that holds 'cells'"
194 (make-row :cells cells)
195
196 (defmethod th ((r row) (c num)) (elt (row-cells r) (num-at c))
197 (defmethod th ((r row) (c sym)) (elt (row-cells r) (sym-at c))
198 (defmethod th ((r row) (n number)) (elt (row-cells r) n))
199
200 ;; ## Cols
201 (defstruct cols all x y klass)
202 (defun cols (lst &aux (l (make-cols)) (at -1))
203 "factory for generating column headers from list of column names"
204 (with-slots (all x y klass) l
205 (dolist (txt lst)
206 (push col all)
207 (when (not (isGoal txt))
208 (if (isGoal txt) (push col y) (push col x)
209 (if (isKlass txt) (setf klass col))))))
210
211 (defmethod add ((i cols) row)
212 "update x and y column headers from data in row. returns row"
213 (dolist (col (cols-x i)) (add col (th row col)))
214 (dolist (col (cols-y i) row) (add col (th row col)))
215
216 ;; ## Data
217 (defstruct data rows cols)
218 (defun data (src &aux (l (make-data)))
219 "create data from either a file called 'src' or a list 'src'"
220 (labels ((update (x) (add i x))
221 (if (stringp src) (with-lines src #'update) (mapc #'update src)
222 i))
223 (defmethod add ((i data) x)
224 "make 'cols' (if currently missing) or update the cols and rows"
225 (with-slots (cols rows) i
226 (if cols
227 (push (add cols (if (row-p x) x (row! x)))
228 rows)
229 (setf cols (cols! x)))))
230
231 (defmethod dist ((i data) (row1 row) (row2 row))
232 "Returns 0..1"
233 (let ((d 0) (n 1E-32))
234 (dolist (col (cols-x (data-cols i))
235 (expt (/ d n) (/ 1 (! p))))
236 (incf d (expt (dist col row1 row2) (! p)))
237 (incf n)))
238
239
240
241
242
243
244

```

```

244 ;; ## Demos
245 (eg "my" (lambda ()
246 "show options"
247 (print 2) t))
248
249 (eg "ls" (lambda ()
250 "show options"
251 (print *settings*) t))
252
253 (eg "geta" (lambda (aux (lst ' (b . 100))))
254 "test adaptive alist"
255 (incf (geta 'a lst))
256 (incf (geta 'a lst))
257 (incf (geta 'a lst))
258 (equal 3 (cdr (assoc 'a lst))))
259
260 (eg "lincs" (lambda ()
261 "testing file reading"
262 (with-lines "/data/mt093.csv"
263 (lambda (s) (format t "~a~%" s))))))
264
265 (eg "num" (lambda (aux (n (num! 10 "num")))
266 "test number"
267 (dolist (i ' (1 1 1 2 2 3)) (add n i)
268 (and (equal 11/7 (mid n)) (equal 0.7867958 (div n))))))
269
270 (eg "sym" (lambda (aux (s (sym! 10 "num")))
271 "test symbols"
272 (dolist (i ' (a a a b b c)) (add s i)
273 (and (equal 'a (mid s)) (equal 1.3787835 (div s))))))
274
275 (eg "cols" (lambda ()
276 "create some columns"
277 (print (cols! ' ("Aas" "state" "Weight-") t)))
278
279 (eg "data" (lambda ()
280 "testing file reading"
281 (print (cols-x (data-cols (data! "/data/mt093.csv")))))
282
283 (setf *settings* (cli (settings *help*)))
284 (if (! help) (about) (egs))

```