


```

1  #!/usr/bin/env lua
2  -- vim : filetype=lua ts=2 sw=2 et :
3  --
4  --
5  -- 
6  --
7  --
8  --
9  --
10 -- (c)2021 Tim Menzies. Permission is hereby granted, free of charge,
11 -- to any person obtaining a copy of this software and associated
12 -- documentation files (the "Software"), to deal in the Software without
13 -- restriction, including without limitation the rights to use, copy,
14 -- modify, merge, publish, distribute, sublicense, and/or sell copies
15 -- of the Software, and to permit persons to whom the Software is
16 -- furnished to do so, subject to the following conditions:
17 --
18 -- The above copyright notice and this permission notice shall be included in all
19 -- copies or substantial portions of the Software.
20 --
21 -- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
22 -- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
23 -- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
24 -- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
25 -- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
26 -- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
27 -- SOFTWARE
28 local help = [[
29 muse [OPTIONS]
30
31 Tree learner (binary splits on numerics using Gaussian approximation)
32 (c)2021 Tim Menzies <timmm@ieee.org> MIT license.
33
34 OPTIONS:
35 -best      X   Best examples are in 1..best*size(all)      = .2
36 -debug     X   run one test, show stackdumps on fail      = pass
37 -epsilon   X   ignore differences under epsilon*stdev     = .35
38 -Far       X   How far to look for remove items          = .9
39 -file      X   Where to read data                         = ../data/auto93.csv
40 -h         X   Show help                                   = false
41 -little    X   size of subset of a list                   = 1024
42 -p         X   distance calc coefficient                  = 2
43 -round     X   Control for rounding numbers               = 2
44 -seed      X   Random number seed;                       = 10019
45 -Stop      X   Create subtrees while at least 2*stop eggs = 4
46 -Tiny      X   Min range size = size(eggs)^tiny          = .5
47 -todo      X   Pass/fail tests to run at start time      = pass
48             If "X=all", then run all.
49             If "X=k" then list all.
50
51 Data read from "-file" is a csv file whose first row contains column
52 names (and the other row contain data. If a name contains ":",
53 that column will get ignored. Otherwise, names starting with upper
54 case denote numerics (and the other columns are symbolic). Names
55 containing "!" are class columns and names containing "+" or "-"
56 are goals to be maximized or minimized.]] --[[
57
58 Internally, columns names are read by a COLS object where numeric,
59 symbolic, and ignored columns generate NUM, SYM, and SKIP instances
60 (respectively). After row1, all the other rows are examples ('EG')
61 which are stored in a SAMPLE. As each example is added to a sample,
62 they are summarized in the COLS' objects.
63
64 Note that SAMPLEs can be created from disk data, or at runtimes from
65 lists of examples (see SAMPLE:clone()). --]]
66
67 local b4={}; for k,_ in pairs(_ENV) do b4[k]=k end
68 local THE = {} -- The THE global stores the global config for this software.
69 -- any line of help text starting with " -" has flag,default as first,last word
70 help:gsub("(^|[-])([^\s%+][^%*%$][^\s%+])", function(flag,x)
71   function(flag,x)
72     for n,word in ipairs(arg) do -- check for any updated to "flag" on command line
73       -- use any command line "word" that matches the start of "flag"
74       if flag:match("^"..word:sub(2).."%") then
75         -- command line "word"s for booleans flip the default value
76         x=(x=="false" and "true") or (x=="true" and "false") or arg[n+1] end end
77       if x=="true" then x=true elseif x=="false" then x=false else x=tonumber(x) or x end
78       THE[flag] = x end)
79
80 THE.seed = THE.seed or 10019
81 if THE.h then return print(help) end

```

```

82 --
83 -- MISC
84 --
85 --
86 -- meta
87 local same
88 function same(x,...) return x end
89
90 -- sorting
91 local push,sort,ones
92 function push(t,x) table.insert(t,x); return x end
93 function sort(t,f) table.sort(t,f); return t end
94 function ones(a,b) return a[1] < b[1] end
95
96 -- tables
97 local copy,keys,map,sum
98 function copy(t, u) u={};for k,v in pairs(t) do u[k]=v end; return u end
99 function keys(t, u) u={};for k,_ in pairs(t) do u[1+#u]=k end; return sort(u) end
100 function map(t,f, u) u={};for k,v in pairs(t) do u[1+#u] =f(k,v) end; return u end
101 function sum(t,f, n) n=0 ;for _,v in pairs(t) do n=n+(f or same)(v) end;return n end
102
103 -- printing utils
104 local hue,shout,out,say,fmt,btw
105 fmt = string.format
106 function say(...) print(string.format(...)) end
107 function btw(...) io.stderr:write(fmt(...).."\n") end
108 function hue(n,s) return string.format("%27[1m27[%sm%27[0m",n,s) end
109 function shout(x) print(out(x)) end
110 function out(t, u,key,val) -- convert nested tables to a string
111   function key(_,k) return string.format("%.5s", k, out(t[k])) end
112   function val(_,v) return out(v) end
113   if type(t) ~= "table" then return tostring(t) end
114   u = #t>0 and map(t, val) or map(keys(t), key)
115   return (t._is or "")..{"..table.concat(u," ").."}" end
116
117 -- reading from file
118 local coerce,csv
119 function coerce(x)
120   if x=="true" then return true elseif x=="false" then return false end
121   return tonumber(x) or x end
122
123 function csv(file, x)
124   file = io.input(file)
125   return function() t,tmp)
126     x = io.read()
127     if x then -- kill space, split on ",", return non-empty lines
128       t={};for y in x:gsub("[\t]*",""):gmatch("[^\t,]+") do push(t,coerce(y)) end
129       if #t>0 then return t end
130     else io.close(file) end end end
131
132 -- maths
133 local log,sqrt,rnd,rnds,roots
134 log = math.log
135 sqrt= math.sqrt
136 function rnd(x,d, n) n=10^(d or THE.round); return math.floor(x*n+0.5) / n end
137 function rnds(t,d)
138   return map(t,function(_,x) return type(x)=="number" and rnd(x,d) or x end) end
139
140 function roots(m1,m2,std1,std2, a,b,c)
141   if std1==std2 then return (m1+m2)/2, (m1+m2)/2 end
142   a = 1/(2*std1^2) - 1/(2*std2^2)
143   b = m2/(std2^2) - m1/(std1^2)
144   c = m1^2 / (2*std1^2) - m2^2 / (2*std2^2) - log(std2/std1)
145   return ((-b - sqrt(b*b - 4*a*c))/(2*a)), ((-b + sqrt(b*b - 4*a*c))/(2*a)) end
146
147 -- random stuff (LUA's built-in randoms give different results on different platfoms)
148 local randi,rand,any,some,shuffle
149 function randi(lo,hi) return math.floor(0.5 + rand(lo,hi)) end
150 function rand(lo,hi)
151   lo, hi = lo or 0, hi or 1
152   THE.seed = (16807 * THE.seed) % 2147483647
153   return lo + (hi-lo) * THE.seed / 2147483647 end
154
155 function any(t) return t[randi(1,#t)] end
156 function some(t,n, u)
157   if n >= #t then return shuffle(copy(t)) end
158   u={}; for i=1,n do push(u,any(t)) end; return u end
159
160 function shuffle(t, j)
161   for i=#t,2,-1 do j=randi(1,i); t[i],t[j]=t[j],t[i] end; return t end
162
163 -- objects
164 local ako,has,obj
165 ako= getmetatable
166 function has(mt,x) return setmetatable(x,mt) end
167 function obj(s, o,new)
168   o = {__is=s, __tostring=out}
169   o.__index = o
170   return setmetatable(o,{__call=function(_,...) return o.new(...) end}) end

```

```

171 --
172 -- NUM
173 --
174 --
175 local NUM=obj"NUM"
176 function NUM.new(inits,at,txt, self)
177   self = has(NUM,{n=0, at=at or 0, txt=txt or "",
178     w=(txt or ""):find "-" and -1 or 1,
179     mu=0, m2=0, lo=math.huge, hi=-math.huge})
180   for _,x in pairs(inits or {}) do self:add(x) end
181   return self end
182
183 -- summarizing
184 function NUM:mid() return self.mu end
185 function NUM:spread() return (self.m2/(self.n-1))^0.5 end
186
187 -- updating
188 function NUM:add(x, d)
189   if x ~= "?" then
190     self.n = self.n + 1
191     d = x - self.mu
192     self.mu = self.mu + d/self.n
193     self.m2 = self.m2 + d*(x-self.mu)
194     self.lo = math.min(x, self.lo)
195     self.hi = math.max(x, self.hi) end
196   return x end
197
198 -- querying
199 function NUM:norm(x, lo,hi)
200   lo,hi = self.lo,self.hi
201   return math.abs(hi - lo) < 1E-9 and 0 or (x-lo)/(hi-lo) end
202
203 function NUM:dist(x,y)
204   if x=="?" then y=self:norm(y); x=y>0.5 and 0 or 1
205   elseif y=="?" then x=self:norm(x); y=x>0.5 and 0 or 1
206   else x, y = self:norm(x), self:norm(y) end
207   return math.abs(x-y) end
208
209 -- discretization
210 function NUM:splits(other, cuts,root1,root2)
211   function cuts(x,s,at) return {
212     {val=x,at=at,txt=fmt("%s <= %s",s,rnd(x)),when=function(z) return z<=x end},
213     {val=x,at=at,txt=fmt("%s > %s",s,rnd(x)),when=function(z) return z > x end}}
214   end
215   root1,root2 = roots(self:mid(), other:mid(), self:spread(), other:spread())
216   if self.mu<=root1 and root1<=other.mu
217   then return cuts(root1,self.txt,self.at)
218   else return cuts(root2,self.txt,self.at) end end
219
220 --
221 -- SYM
222 --
223 local SYM=obj"SYM"
224 function SYM.new(inits,at,txt, self)
225   self= has(SYM,{n=0, at=at or 0, txt=txt or "",
226     seen={}, mode=nil, most=0})
227   for _,x in pairs(inits or {}) do self:add(x) end
228   return self end
229
230 -- Summarizing
231 function SYM:mid() return self.mode end
232 function SYM:spread()
233   return sum(self.seen, function(n) return -n/self.n*log(n/self.n,2) end) end
234
235 -- update
236 function SYM:add(x)
237   if x ~= "?" then
238     self.n = 1 + self.n
239     self.seen[x] = (self.seen[x] or 0) + 1
240     if self.seen[x] > self.most then self.mode, self.most = x, self.seen[x] end
241     return x end end
242
243 -- querying
244 function SYM:dist(x,y) return x==y and 0 or 1 end
245
246 -- discretization
247 function SYM:splits(other, cut,out)
248   function cut(_,x) return
249     {val=x, at=self.at, txt=fmt("%s==%s",self.txt,x),
250     when = function(z) return z==x end} end
251   out={}
252   for k,_ in pairs(self.seen) do out[k]=k end
253   for k,_ in pairs(other.seen) do out[k]=k end
254   return map(sort(out),cut) end
255

```

```

256 --
257 -- SKIP
258 --
259 --
260 -- Columns for values we want to ignore.
261 local SKIP=obj"SKIP"
262 function SKIP.new(inits,at,txt)
263   return has(SKIP,{n=0, at=at or 0, txt=txt or ""}) end
264
265 function SKIP:mid() return "?" end
266 function SKIP:spread() return 0 end
267 function SKIP:add(x) return x end
268 function SKIP:splits(_) return {} end
269
270 --
271 -- COLS
272 --
273 -- Convert column headers into NUMs and SYMs, etc.
274 local COLS=obj"COLS"
275 function COLS.new(names, self, new,what)
276   self = has(COLS, {names=names, xs={}, all={}, ys={}})
277   for n,x in pairs(names) do
278     new = (x:find"." and SKIP or x:match"^[A-Z]" and NUM or SYM) ({},n,x)
279     push(self.all, new)
280     if not x:find"." then
281       if x:find"|" then self.klass = new end
282       what = (x:find "-" or x:find "+") and "ys" or "xs"
283       push(self[what], new) end end
284   return self end
285
286 -- Updates
287 function COLS:add(eg)
288   return map(eg, function(n,x) self.all[n]:add(x); return x end) end
289

```

```

290 --
291 -- EG
292 --
293 --
294 -- One example
295 local EG=obj"EG"
296 function EG.new(cells) return has(EG,{cells=cells}) end
297
298 -- Summarizing
299 function EG:cols(all)
300   return map(all,function(_,c) return self.cells[c.at] end) end
301
302 -- Queries
303 function EG:dist(other,cols, a,b,d,n,inc)
304   d,n = 0,0
305   for _,col in pairs(cols) do
306     a,b = self.cells[col.at], other.cells[col.at]
307     inc = a=="?" and b=="?" and 1 or col:dist(a,b)
308     d = d + inc^THE.p
309     n = n + 1 end
310   return (d/n)^(1/THE.p) end
311
312 -- Sorting
313 function EG:better(other,cols, e,n,a,b,s1,s2)
314   n,s1,s2,e = #cols, 0, 0, 2.71828
315   for _,num in pairs(cols) do
316     a = num:norm(self.cells[num.at])
317     b = num:norm(other.cells[num.at])
318     s1 = s1 - e^(num.w * (a-b)/n)
319     s2 = s2 - e^(num.w * (b-a)/n) end
320   return s1/n < s2/n end
321
322 -- SAMPLE
323 --
324 --
325 -- SAMPLEs hold many examples
326 local SAMPLE=obj"SAMPLE"
327 function SAMPLE.new(inits, self)
328   self = has(SAMPLE,{cols=nil, eggs={}})
329   if type(inits)=="string" then for eg in csv(inits) do self:add(eg) end end
330   if type(inits)=="table" then for eg in pairs(inits) do self:add(eg) end end
331   return self end
332
333 -- Create a new sample with the same structure as this one
334 function SAMPLE:clone(inits, out)
335   out = SAMPLE.new()
336   out:add(self.cols.names)
337   for _,eg in pairs(inits or {}) do out:add(eg) end
338   return out end
339
340 -- Updates
341 function SAMPLE:add(eg)
342   eg = eg.cells and eg.cells or eg
343   if self.cols
344     then push(self.egs, EG(eg)); self.cols:add(eg)
345     else self.cols = COLS(eg) end end
346
347 -- Distance queries
348 function SAMPLE:neighbors(eg1,egs,cols, dist_eg2)
349   dist_eg2 = function(_,eg2) return {eg1:dist(eg2,cols or self.cols.xs),eg2} end
350   return sort(map(egs or self.egs,dist_eg2),ones) end
351
352 function SAMPLE:distance_farExample(eg1,egs,cols, tmp)
353   tmp = self:neighbors(eg1, egs, cols)
354   return table.unpack(tmp[#tmp*THE.Far//1]) end
355
356 -- Discretization
357 -- need to uthe super
358 function SAMPLE:twain(egs,cols, _,egs,north,south,a,b,c,lo,hi)
359   _,north = self:distance_farExample(any(egs), egs, cols)
360   c,south = self:distance_farExample(north, egs, cols)
361   for _,eg in pairs(egs) do
362     a = eg:dist(north, cols)
363     b = eg:dist(south, cols)
364     eg.x = (a^2 + c^2 - b^2)/(2*c) end
365   lo, hi = self:clone(), self:clone()
366   for n,eg in pairs(sort(egs, function(a,b) return a.x < b.x end)) do
367     if n < .5*#egs then lo:add(eg) else hi:add(eg) end end
368   return lo, hi end
369
370 function SAMPLE:mid(cols)
371   return map(cols or self.cols.all,function(_,col) return col:mid() end) end
372
373 function SAMPLE:spread(cols)
374   return map(cols or self.cols.all,function(_,col) return col:spread() end) end
375
376 function SAMPLE:sorted()
377   self.egs= sort(self.egs, function(eg1,eg2) return eg1:better(eg2,self.cols.ys) end)
378   return self.egs end
379

```

```

380 --
381 -- SAMPLE TREE
382 --
383 --
384 -- need to sort first
385
386 -- how to score
387 function SAMPLE:splits(other,both, cuts,unplaced,place,score)
388   function guess(todos,cuts, f)
389     for _,todo in pairs(todos) do
390       f=function(_,cut)
391         return (Row(cut.has:mid()):dist(todo, both.cols.xs),cut) end
392       sort(map(cuts,f),ones)[1][2].has:add(todo) end
393     return cuts end
394   function divide(cuts, todos,placed)
395     todos = {}
396     for _,eg in pairs(both.egs) do
397       placed = false
398       for _,cut in pairs(cuts) do
399         if cut.what(eg.cells[cut.at])
400           then cut.has = cut.has or self.clone()
401               cut.has:add(eg)
402               placed = true
403               break end end
404       if not placed then push(todos, eg) end end
405     return guess(todos,cuts) end
406   function score(cut, m,n)
407     m,n = #cut.has.egs,both.egs; return -m/n*log(m/n,2) end
408   local best, cutsx, tmp = math.huge
409   for pos,col in pairs(both.cols.xs) do
410     cutsx = col:splits(other.cols.xs[pos])
411     tmp = sum(divide(cutsx),score)
412     if tmp < best then best,cuts = tmp,cutsx end end
413   return cuts end
414
415 function SAMPLE:tree(top)
416   top = top or self
417   one,two = self:twain(self.egs, top.cols.xs)
418   for _,cut in pairs(one:splits(two,self)) do
419     if cut.stats.n > (#top.egs)^THE.Tiny then
420       cut.sub= cut.has:tree(top) end end end
421
422 function SAMPLE:show(tree, vals,showl)
423   vals=function(a,b) return a.val < b.val end
424   function showl(tree,pre)
425     if #tree.kids==0 then io.write(fmt("==> %s[%s]",tree.mode, tree.n)) end
426     for _,kid in pairs(sort(tree.kids,vals)) do
427       io.write("\n"..fmt("%s%s",pre, showDiv(i, kid.at, kid.val)))
428       showl(kid.sub, pre.."|..") end
429   end -----
430   showl(tree,""); print("") end
431

```

```

432 -----
433
434 -- EXAMPLES
435 --
436 --
437 local go={}
438 function go.ls()
439   print("\nlua"..arg[0].." -todo ACTION\n\nACTIONS:")
440   for _,k in pairs(keys(go)) do print("-todo",k) end end
441
442 function go.pass() return true end
443 function go.the() shout(THE) end
444 function go.bad( s) assert(false) end
445
446 function go.sort( u,t)
447   t={}; for i=100,1,-1 do push(t,i) end
448   t=sort(t,function(x,y)
449     if x+y<20 then return x>y else return x<y end end)
450   assert(sum(t,function(x) return x*100 end)==505000)
451   assert(t[1] == 10)
452   assert(t[#t]==100)
453   u=copy(t)
454   t[1] = 99
455   assert(u[1] ~= 99) end
456
457 function go.out( s)
458   assert({"age 21 :milestones {1 2 3 4} :name tim"}==out(
459     {name='tim', age=21, milestones={1,2,3,4}}))end
460
461 function go.file( n)
462   for _,t in pairs({"true",true,"boolean"}, {"false",false,"boolean"},
463     {"42.1",42.1,"number"}, {"32zz","32zz","string"},
464     {"nil","nil","string"}} do
465     assert(coerce(t[1])==t[2])
466     assert(type(coerce(t[1]))==t[3]) end
467   n =0
468   for row in csv(THE.file) do
469     n = n + 1
470     assert(#row==8)
471     assert(n==1 or type(row[1])=="number")
472     assert(n==1 or type(row[8])=="number") end end
473
474 function go.rand( t,u)
475   t,u={},{}; for i=1,20 do push(u,push(t,100*rand())) end
476   t= sort(rnds(t,0))
477   assert(t[1]==3 and t[#t]==88)
478   t= sort(some(t,4))
479   assert(#t==4)
480   assert(t[1]==7)
481   assert(79.5 == rnds(shuffle(u))[1])
482 end
483
484 function go.num( cut,min, z,r1,r1,x,y)
485   z = NUM(9,2,5,4,12,7,8,11,9,3,7,4,12,5,4,10,9,6,9,4)
486   assert(7 == z:mid(), 3.06 == rnd(z:spread(),2))
487   r1,r2 = roots(2.5, 5, 1.1, .9)
488   assert(rnd(r2,2)==3.8)
489   x, y = NUM(), NUM()
490   for i=1,20 do x:add(rand(1,5)) end
491   for i=1,20 do y:add(randi(20,30)) end
492   for _,cut in pairs(x:splits(y)) do shout(cut) end end
493
494 function go.sym( cut,min,w,z)
495   w = SYM{"m","m","m","m","b","b","c"}
496   z = SYM{"a","a","a","a","b","b","c"}
497   assert(1.38 == rnd(z:spread(),2))
498   for _,cut in pairs(w:splits(z)) do shout(cut) end end
499
500 function go.sample( s,egs,xs,ys,scopy)
501   s=SAMPLE(THE.file)
502   scopy=s:clone(s.egs)
503   print(s.cols.all[1]:spread(), scopy.cols.all[1]:spread())
504   xs,ys= s.cols.xs, s.cols.ys
505   assert(4 == #xs)
506   assert(3 == #ys)
507   egs=s:sorted()
508   shout(rnds(s:mid(ys),1));
509   shout(rnds(map(s:spread(ys),function(_,x) return .35*x end), 1)); print("")
510   for i=1,10 do shout(rnds(egs[i]:cols(ys),1)) end; print("")
511   for i=#egs,#egs-10,-1 do shout(rnds(egs[i]:cols(ys),1)) end end
512
513 function go.dist( s,xs,sorted, show )
514   s=SAMPLE(THE.file)
515   xs= s.cols.xs
516   for k,eg2 in pairs(s.egs) do
517     if k > 20 then break end
518     print(s.egs[1]:dist(eg2, xs)) end
519   sorted = s:neighbors(s.egs[1], s.egs,xs)
520   show=function(i)print(rnd(sorted[i][1],2), out(sorted[i][2]:cols(xs))) end
521   for i=1,10 do show(i) end; print("")

```

```

522   for i=#sorted-10,#sorted do show(i) end end
523
524 function go.far( s,xs,d,eg2)
525   s = SAMPLE(THE.file)
526   xs = s.cols.xs
527   for k,eg1 in pairs(shuffle(s.egs)) do
528     if k > 10 then break end
529     d,eg2 = s:distance_farExample(eg1, s.egs, xs)
530     print(rnd(d), out(eg1:cols(xs)), out(eg2:cols(xs))) end end
531
532 function go.twain( s,lo,hi)
533   s = SAMPLE(THE.file)
534   lo,hi = s:twain(s.egs, s.cols.xs)
535   print(#lo.egs, #hi.egs)
536 end
537
538 function go.splits( lo,hi)
539   lo,hi = SAMPLE(THE.file):twain()
540   print(#lo.egs, #hi.egs)
541 end
542
543 function go.kordered( s,n)
544   s = ordered(slurp())
545   n = #s.egs
546   shout(s.heads)
547   for i=1,15 do shout(s.egs[i].cells) end
548   print("#")
549   for i=n,n-15,-1 do shout(s.egs[i].cells) end end
550
551 function go.ksymcuts( s,xpect,cuts)
552   s=ordered(slurp())
553   print(out(s.cols.xs),out(s.cols.ys))
554   xpect,cuts = symcuts(7,s.egs, "origin")
555   for _,cut in pairs(cuts) do print(xpect, out(cut)) end end
556
557 function go.knumcuts( s,xpect,cuts)
558   s=ordered(slurp())
559   xpect,cuts = numcuts(s,2,s.egs,"Displcment")
560   if xpect then
561     for _,cut in pairs(cuts) do print(xpect, out(cut)) end end end
562
563 function go.katcuts(s,cuts,at,ynum)
564   s=ordered(slurp())
565   ynum=NUM(a); map(s.egs,function(_,eg) add(ynum, eg.klass) end)
566   at,cuts = at_cuts(s,egs,sd(ynum)*THE.epsilon, (#s.egs)^THE.Tiny)
567   for _,cut in pairs(cuts) do print(at, out(cut)) end end
568
569 --
570 -- START-UP
571 --
572 local fails,defaults,todos,ok,msg
573 fails, defaults = 0, copy(THE)
574 go[ THE.debug ]()
575
576 todos = THE.todo == "all" and keys(go) or {THE.todo}
577 for _,todo in pairs(todos) do
578   THE = copy(defaults)
579   ok,msg = pcall( go[todo] )
580   if ok then btw("%s%",hue(32,"--PASS"),todo)
581     else btw("%s%s%",hue(31,"--FAIL"),todo,msg); fails=fails+1 end end
582
583 btw(hue(33,"-- %s errors"),fails)
584 for k,v in pairs(_ENV) do
585   if not b4[k] then btw(hue(31,"-- rogue? %s %s"),k,type(v)) end end
586 os.exit(fails)

```