

```

1 module My
2 using Parameters
3 using Random
4
5 @with_kw mutable struct Config
6     char = (skip="?", less=">", more="<", num="$", klass="!")
7     str = (skip="?")
8     some = (max=512, step=.5, cohen=.3, trivial=1.05)
9     seed = 1
10 end
11
12 THE = Config()
13 Random.seed!(THE.seed)
14
15 # -----
16 same(s) = s
17 int(x) = floor{Int, x}
18 any(a) = a[ int(size(a) * rand()) + 1 ]
19
20 function say(i)
21     s_pre = "$(typeof(i)){"
22     for f in sort!([x for x in fieldnames(typeof(i))
23                     if !("$x"[1] == '_')])
24         g = getfield(i, f)
25         s = s * pre * "$f=$g"
26         pre = ", "
27     end
28     print(s * "}")
29 end
30
31 # -----
32 add!(init=[], i=Some) = incs!(i(), init, 1)
33 subs!(init=[], i=Some) = incs!(i(), init, -1)
34 add!(i, x) = inc!(i, x, 1)
35 sub!(i, x) = inc!(i, x, -1)
36
37 # my cols can do:
38 # incs!, incl, stale!, fresh, mid, var
39 # my cols know about:
40 # w, pos, txt, w, key, n
41 incs!(i, init=[], w=1) = begin incl!(i, x, w) for x in init; i end
42
43 function incl!(i, x, w=1)
44     y = i.key(x)
45     if y != THE.str.skip
46         stale(i)
47         i.n += w
48         incl!(i, y, w) end
49 end
50
51 # -----
52 @with_kw mutable struct Some
53     pos=0; txt=""; w=1; key=same; n=0;
54     lo=10^32; hi=-1*10^32; mu=0; m2=0; sd=nothing end
55
56 mid(i::Num) = i.mu
57 stale(i::Num) = i.sd == nothing
58
59 function var(i::Num)
60     if i.sd == nothing
61         i.sd = i.n < 2 ? 0 : (i.m2 / (i.n - 1 + 10^-32))^0.5 end
62     i.sd
63 end
64
65 function incl!(i::Num, x)
66     i.lo = min(i.lo, x)
67     i.hi = max(i.hi, x)
68     d = x - i.mu
69     i.mu += d / i.n
70     i.m2 += d * (x - i.mu)
71 end
72
73 # -----
74
75 @with_kw mutable struct Some
76     pos=0; txt=""; w=1; key=same; n=0;
77     all=[]; max=THE.some.max; tidy=false end
78
79 p(i::Some, n) = begin fresh(i); i.all[ int(n*length(i.all))+1 ]
80 end
81
82 stale(i::Some) = i.tidy=false
83
84 function fresh(i::Some)
85     if !i.tidy
86         sort!(i.all)
87         i.tidy=true end end
88
89 has(i::Some, n) = begin fresh(i); i.all[n] end
90 mid(i::Some, lo=1, hi=length(i.all)) = has(i, int((lo+hi-lo)*.5))
91
92 function var(i::Some, lo=1, hi=length(i.all))
93     fresh(i)
94     n10 = int((lo+hi-lo)*.1) + 1
95     n90 = int((lo+hi-lo)*.9) + 1
96     (i.all[n90] - i.all[n10])/2.7
97 end
98
99 function incl!(i::Some, x, w=1)
100     m = length(i.all)
101     if m < i.max
102         push!(i.all, x)
103     elseif rand() < m/i.n
104         i.all[ int(m*rand()) + 1 ] = x end
105 end
106
107 "If i.all is broken at the points listed in "a"
108 between "lo" and "hi", what is the expected value?"
109 function xpect(i::Some, a, lo=1, hi=length(i.all))
110     el(x, y) = (y-x+1)/(hi-lo+1)*var(i, x, y)
111     e, m = 0, lo
112     for n in a
113         e += el(m, n)
114         m = n+1
115     end
116     e + el(m, hi)
117 end
118
119 div(i::Some) = begin fresh(i); div(i.all, i.key) end
120
121 # -----
122 @with_kw mutable struct Range
123     lo=0; hi=0; _all=[]; start=0; stop=0; w=0; _kids=[] end
124
125 Base.show(io::IO, i::Range) = say(i)
126
127 "assumes lst is sorted"
128 function div(lst::Array, key=same)
129     the = THE.some
130     x(z) = key(lst[int(z)])
131     val(y, z, p=0.5) = x(y+(z-y)*p)
132     var(y, z) = (val(y, z, 0.9) - val(y, z, 0.1))/2.7
133     function xchop(lo, hi, out=nothing)
134         best = var(lo, hi)
135         for j = lo+step:hi-step
136             now, after = x(j), x(j+1)
137             if now != after
138                 if after - start > epsilon
139                     if stop - now > epsilon
140                         if abs(val(lo, j) - val(j+1, hi)) > epsilon
141                             n1, n2 = j-lo+1, hi-j
142                             here = (var(lo, j)*n1 + var(j+1, hi)*n2)/(n1+n2)
143                             if here<the.trivial < best
144                                 best, out = here, j end end end end end end
145                             return out
146                         end
147                     function xchops(lo, hi, ranges, cut = chop(lo, hi))
148                         if cut == nothing
149                             push!(ranges, Range(lo=x(lo), hi=x(hi),
150                                 _all=lst[lo:hi], start=lo, stop=hi))
151                         else
152                             xchops(lo, cut, ranges)
153                             xchops(cut+1, hi, ranges) end
154                     end
155                     n = length(lst)
156                     epsilon = var(1, n) * the.cohen
157                     step, start, stop = int(n*the.step)-1, x(1), x(n)
158                     xchops(1, n, [])
159                 end
160                 function chops(lo, hi, ranges, chop)
161                     cut = chop(lo, hi)
162                     if cut == nothing
163                         push!(ranges, Range(lo=x(lo), hi=x(hi),
164                             _all=lst[lo:hi], start=lo, stop=hi))
165                     else
166                         ychops(lo, cut, ranges)
167                         ychops(cut+1, hi, ranges) end
168                     end
169                 end
170                 # function unite(rs, y=same, better=<, yis=Num)
171                 #     the = THE.some
172                 #     all(x=yis(key=y), a=[])= begin [incs!(x, r._all) for r in a]; x
173                 #         end
174                 #     function ychop(lo, hi, best, rs, out=nothing)
175                 #         left = yis(key=y)
176                 #         for j in lo:hi-1
177                 #             l = all(x=left, [rs[j]])
178                 #             r = all(a=rs[j+1:hi])
179                 #             now = (var(l)*l.n + var(r)*r.n)/(l.n + r.n)
180                 #             if better(now, the.trivial, best)
181                 #                 best, out = now, j end end
182                 #             out
183                 #         end
184                 #     end
185                 #     f = (start, stop) -> ychop(start, stop, )
186                 #     chop(1, length(rs), [], var(all(ranges)))
187
188 #end
189 # -----
190 @with_kw mutable struct Sym
191     pos=0; txt=""; w=1; key=same; n=0;
192     seen=Dict(); mode=nothing; ent=nothing; end
193
194 mid(i::Sym) = begin i.fresh(); i.mode end
195 var(i::Sym) = begin i.fresh(); i.ent end
196
197 stale(i::Sym) = i.mode, i.ent = nothing, nothing
198 function fresh(i::Sym)
199     if i.mode == nothing
200         i.ent, most = 0, 0
201         for (k, n) in i.seen
202             p = n/i.n
203             i.ent -= p*log(2, p)
204             if n > most
205                 most, i.mode = n, k end end end
206         end
207     end
208     function incl!(i::Sym, x, w=1)
209         new = w + (haskey(i.seen, x) ? i.seen[x] : 0)
210         i.seen[x] = max(new, 0)
211     end
212
213 # -----
214 norm(i::Sym, x) = x
215 norm(i::Some, x) = begin fresh(i); (x-i.all[1])/((i.all[end]-i.all[1]) end
216
217 difference(i::Sym, x, y) = x==THE.string.skip ? 1 : x == y
218 function difference(i::Some, x, y, no = THE.string.skip)
219     d(a, b) = begin a = norm(i, a); b = a<0.5 ? 1 : 0; abs(a-b) end
220     if x=no && y=no
221         elseif x=no d(y, x)
222         elseif y=no d(x, y)
223         else
224             abs(norm(i, x) - norm(i, y)) end
225         end
226     end
227
228 # -----
229 @with_kw struct Lines file; src=open(file) end
230
231 "Define an iterator that returns a comma-seperated file, one
232 record at a time without loading the whole file into memory."
233 function Base.iterate(it::Lines, (n, want)=(1, []))
234     "Split on comma, coerce strings to numbers or strings, as
235     appropriate."
236     coerce(s) = map(coerce1, split(s, ","))
237     coerce1(s) = ((x = tryparse(Float64, s))=nothing) ? s : x
238
239     "Coerce strings. If first row, check what columns we should
240     use.
241     Only return those columns."
242     function cols(a)
243         if n == 1
244             want = [i for (i, s) in enumerate(a) if !('?' in s)] end
245             [a[i] for i in want]
246         end
247
248     "Delete comments and whitespaces. Lines ending in
249     ',' are joined to the next. Skip empty lines."
250     function row(txt="")
251         while true
252             if eof(it.src) return txt end
253             new = readline(it.src)
254             new = replace(new, r"([ \t\n]#.*)"=>"")
255             if sizeof(new) != 0
256                 txt *= new
257                 if txt[end] != ','
258                     return txt end end end
259
260         new = row()
261         if sizeof(new) > 0
262             (n, cols(coerce(new))) , (n+1, want) end
263         end
264     end
265
266 # -----
267 id=0
268
269 @with_kw mutable struct Tbl
270     rows=[]; cols=Cols() end
271
272 @with_kw mutable struct Row
273     cells=[]; cooked=[]; id=global id+= 1
274 end
275
276 say(Row())
277 say(Row())
278
279 @with_kw mutable struct Cols
280     x = (all=[],, nums=[],, syms=[]])
281     y = (all=[],, nums=[],, syms=[],, goals=[]])
282     klass=""
283     all = []; nums = []; syms = []; end
284
285 function table(file::String)
286     t=Tbl()
287     for (n, a) in Lines(file=file)
288         n==1 ? head!(t, a) : row!(t, a) end
289     end
290
291 function row!(i::Tbl, a)
292     [add!(c, a[c.pos]) for c in i.cols.all]
293     push!(i.rows, Row(cells=a))
294 end
295
296 head!(i::Tbl, a) = [head!(i.cols, n, x) for (n, x) in enumerate(a)]
297
298 function head!(i::Cols, n, txt)
299     the = THE.char
300     goalp() = the.less in txt || the.more in txt
301     nump() = the.num in txt || goalp()
302     yp() = klassp() || goalp()
303     klassp() = the.klass in txt
304     x = nump() ? Some : Sym
305     y = x(pos=n, txt=txt)
306     if klassp() i.klass = y end
307     if goalp() push!(i.y.goals, y) end
308     if nump()
309         push!(i.nums, y); push!(yp() ? i.y.nums : i.x.nums, y)
310     else
311         push!(i.syms, y); push!(yp() ? i.y.syms : i.x.syms, y)
312     end
313     push!(yp() ? i.y.all : i.x.all, y)
314     push!(i.all, y)
315 end
316
317 # -----
318 function tbl1(f="data/auto.csv")
319     t = table(f)
320     println("n ", length(t.rows))
321     for col in t.cols.x.nums
322         println(div(col)) #println(var(col), " ", col.all)
323     end
324 end
325
326 function nums(f="data/auto.csv")
327     t = table(f)
328     #println(t.rows[end].cells)
329     for num in t.cols.x.nums
330         d=div(num)
331         println(num.txt, " ", length(d))
332         println(d)
333     end
334 end
335
336 function sym1()
337     s=Sym()
338     [add!(s, x) for x in "aaaabbc"]
339 end
340
341 function Lines1(f="data/weather.csv")
342     m=1
343     print(m)
344     for (n, tmp) in Lines(file= f)
345         m += sizeof(tmp) #println(n, " ", tmp)
346         if mod(n, 1000) == 0 println(n, ":", m) end
347     end
348     print(m)
349 end
350
351 function num1(x)
352     if x<0.3 return 0.1 end
353     if x<0.7 return 0.8 end
354     return 0.9
355 end
356
357 function numbers1(s=Some())
358     [add!(s, num1(rand())) for i in 1:100]
359     println([has(s, i) for i in div(s)])
360 end
361
362 function numbers2(n=2, s=Some())
363     [add!(s, rand()*0.5) for i in 1:10^n]
364     println([i, has(s, i)] for i in div(s)])
365 end
366
367 #some1()
368 #sym1()
369 #time tbl1("data/xomo10000.csv")
370 #time tbl1("data/weather.csv")
371 #time nums("data/xomo10000.csv")
372 #numbers1()
373 #time numbers2(3)
374 end

```