

```

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:
33: adds!(init=[], i=Some) = incs!(i(), init, 1)
34: subs!(init=[], i=Some) = incs!(i(), init, -1)
35: add!(i,x) = inc!(i , x, 1)
36: sub!(i,x) = inc!(i , x, -1)
37:
38: # my cols can do:
39: # incs!, inc!, statel, fresh, mid, var
40: # my cols know about:
41: # w,pos,txt,w,key,n
42: incs!(i,init=[],w=1) = begin [inc!(i,x,w) for x in init]; i end
43:
44: function inc!(i,x,w=1)
45:   y=i.key(x)
46:   if y != THE.str.skip
47:     stale(i)
48:     i.n += w
49:     incl!(i, y,w) end
50: end
51:
52: # -----
53: @with_kw mutable struct Num
54:   pos=0; txt=""; w=1; key=same; n=0;
55:   lo=10^32; hi=-1*10^32; mu=0; m2=0; sd=nothing end
56:
57: mid(i::Num) = i.mu
58: stale(i::Num) = i.sd == nothing
59:
60: function var(i::Num)
61:   if i.sd == nothing
62:     i.sd = i.n < 2 ? 0 : (i.m2 / (i.n - 1 + 10^-32))^0.5 end
63:   i.sd
64: end
65:
66: function incl!(i::Num,x)
67:   i.lo = min(i.lo, x)
68:   i.hi = max(i.hi, x)
69:   d = x - i.mu
70:   i.mu += d / i.n
71:   i.m2 += d * (x - i.mu)
72: end
73:
74: # -----
75:
76: @with_kw mutable struct Some
77:   pos=0; txt=""; w=1; key=same; n=0;
78:   all=[]; max=THE.some.max ;tidy=false end
79:
80: p(i::Some,n) = begin fresh(i); i.all[ int(n*length(i.all))+1 ] end
81: stale(i::Some) = i.tidy=false
82:
83: function fresh(i::Some)
84:   if !i.tidy
85:     sort!(i.all)
86:     i.tidy=true end end
87:
88: has(i::Some,n) = begin fresh(i); i.all[n] end
89: mid(i::Some, lo=1, hi=length(i.all)) = has(i, int(lo+(hi-lo)*.5))
90:
91: function var(i::Some, lo=1, hi=length(i.all))
92:   fresh(i)
93:   n10 = int(lo+(hi-lo)*.1) + 1
94:   n90 = int(lo+(hi-lo)*.9) + 1
95:   (i.all[n90] - i.all[n10])/2.7
96: end
97:
98: function incl!(i::Some, x,w=1)
99:   m = length(i.all)
100:   if m < i.max
101:     push!(i.all,x)
102:     elseif rand() < m/i.n
103:     i.all[ int(m*rand()) + 1 ] = x end
104: end
105:
106: "If i.all is broken at the points listed in 'a'
107: between 'lo' and 'hi', what is the expected value?"
108: function xpect(i::Some,a,lo=1,hi=length(i.all))
109:   e1(x,y) = (y-x+1)/(hi-lo+1)*var(i,x,y)
110:   e,m = 0,lo
111:   for n in a
112:     e += e1(m,n)
113:     m = n+1
114:   end
115:   e + e1(m,hi)
116: end
117:
118: div(i::Some) = begin fresh(i); div(i.all,i.key) end
119:
120: # -----
121: @with_kw mutable struct Range
122:   lo=0; hi=0; _all=[]; start=0; stop=0; w=0; _kids=[] end
123:
124: Base.show(io::IO, i::Range) = say(i)
125:
126: "assumes lst is sorted"
127: function div(lst::Array, key=same)
128:   the = THE.some
129:   x(z) = key(lst[ int(z) ])
130:   val(y,z,p=0.5) = x(y+(z-y)*p)
131:   var(y,z) = (val(y,z,0.9) - val(y,z,0.1))/2.7
132:   function xchop(lo,hi,out=nothing)
133:     best = var(lo,hi)
134:     for j = lo+step:hi-step
135:       now, after = x(j), x(j+1)
136:       if now != after
137:         if after - start > epsilon
138:           if stop - now > epsilon
139:             if abs(val(lo,j) - val(j+1,hi)) > epsilon
140:               n1,n2 = j-lo+1, hi-j
141:               here = (var(lo,j)*n1 + var(j+1,hi)*n2)/(n1+n2)
142:               if here*the.trivial < best
143:                 best,out = here,j end end end end end
144:               return out
145:             end
146:             function xchops(lo,hi,ranges, cut = chop(lo,hi))
147:               if cut == nothing
148:                 push!(ranges, Range(lo=x(lo), hi=x(hi),
149:                   _all=lst[lo:hi],start=lo,stop=hi))
150:               else
151:                 xchops(lo, cut, ranges)
152:                 xchops(cut+1, hi, ranges) end
153:             end
154:             #-----
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:
161:           function chops(lo,hi,ranges,chop)
162:             cut = chop(lo,hi)
163:             if cut == nothing
164:               push!(ranges, Range(lo=x(lo), hi=x(hi),
165:                 _all=lst[lo:hi],start=lo,stop=hi))
166:             else
167:               ychops(lo, cut, ranges)
168:               ychops(cut+1, hi, ranges) end
169:           end
170:         end
171:
172:         #function unite(rs, y=same,better= <, yis=Num)
173:         # the = THE.some
174:         # all(x=yis(Key=y),a=[])= begin [incs!(x,r._all) for r in a]; x end
175:         # function ychop(lo,hi,best,rs,out=nothing)
176:         # left = yis(Key=y)
177:         # for j in lo:hi-1
178:         #   l= all(x=left,[rs[j]])
179:         #   rall(a=rs[j+1:hi])
180:         #   now = (var(l)*1.n + var(r)*r.n)/(1.n + r.n)
181:         #   if better(now*the.trivial, best)
182:         #     best,out = now,j end end
183:         #   out
184:         #   end
185:         #   f = (start,stop) -> ychop(start,stop,r)
186:         #   chop(1,length(rs),[], var(all(ranges)))
187:         # end
188:         # -----
189:         @with_kw mutable struct Sym
190:           pos=0; txt=""; w=1; key=same; n=0;
191:           seen=Dict{ }; mode=nothing; ent=nothing; end
192:
193:           mid(i::Sym) = begin i.fresh(); i.mode end
194:           var(i::Sym) = begin i.fresh(); i.ent end
195:
196:           stale(i::Sym) = i.mode,i.ent = nothing,nothing
197:           function fresh(i::Sym)
198:             if i.mode == nothing
199:               i.ent, most = 0,0
200:               for (k,n) in i.seen
201:                 p = n/i.n
202:                 i.ent -= p*log(2,p)
203:                 if n > most most,i.mode = n.k end end end
204:             end
205:
206:             function incl!(i::Sym,x,w=1)
207:               new = w + (haskey(i.seen, x) ? i.seen[x] : 0)
208:               i.seen[x] = max(new,0)
209:             end
210:
211:             # -----
212:             norm(i::Sym, x) = x
213:             norm(i::Some,x) = begin fresh(i); (x-i.all[1])/(i.all[end]-i.all[1]) end
214:
215:             difference(i::Sym, x,y) = x==THE.string.skip ? 1 : x == y
216:             function difference(i::Some,x,y, no = THE.string.skip)
217:               d(a,b) = begin a = norm(i,a); b = a<0.5 ? 1 : 0; abs(a-b) end
218:               if x=no && y=no 1
219:                 elseif x=no d(y,x)
220:                 elseif y=no d(x,y)
221:                 else abs(norm(i,x) - norm(i,y)) end
222:             end
223:
224:             # -----
225:             @with_kw struct Lines file; src=open(file) end
226:
227:             "Define an iterator that returns a comma-seperated file, one
228:             record at a time without loading the whole file into memory."
229:             function Base.iterate(it::Lines, (n,want)=(1,[]))
230:               "Split on comma, coerce strings to numbers or strings, as appropriate."
231:               coerce(s) = map(coerce!, split(s,","))
232:               coercel(s) = ((x = tryparse(Float64,s))==nothing) ? s : x
233:
234:               "Coerce strings. If first row, check what columns we should use.
235:               Only return those columns."
236:               function cols(a)
237:                 if n == 1
238:                   want = [i for (i,s) in enumerate(a) if !{'?' in s}] end
239:                 [a[i] for i in want]
240:               end
241:
242:               "Delete comments and whitespace. Lines ending in
243:               ', ' are joined to the next. Skip empty lines."
244:               function row(txt="")
245:                 while true
246:                   if eof(it.src) return txt end
247:                   new = readline(it.src)
248:                   new = replace(new, r"([\t\n]|#.*)">""))
249:                   if sizeof(new) != 0
250:                     txt *= new
251:                     if txt[end] != ', '
252:                       return txt end end end end
253:
254:                   new = row()
255:                   if sizeof(new) > 0
256:                     (n, cols(coerce(new))), (n+1,want) end
257:                   end
258:
259:                   #-----
260:                   id=0
261:
262:                   @with_kw mutable struct Tbl
263:                     rows=[]; cols=Cols() end
264:
265:                     @with_kw mutable struct Row
266:                       cells=[]; cooked=[]; id=global id+= 1
267:                     end
268:
269:                     270:

```

```

271: say(Row())
272: say(Row())
273: @with_kw mutable struct Cols
274:   x = (all=[], nums=[], syms=[])
275:   y = (all=[], nums=[], syms=[], goals=[])
276:   klass=""
277:   all = []; nums = []; syms = []; end
278:
279: function table(file::String)
280:   t=Tbl()
281:   for (n,a) in Lines(file=file)
282:     n==1 ? head!(t,a) : row!(t,a) end
283:   t
284: end
285:
286: function row!(i::Tbl,a)
287:   [add!(c,a[c.pos]) for c in i.cols.all]
288:   push!(i.rows, Row(cells=a) )
289: end
290:
291: head!(i::Tbl,a) = [head!(i.cols,n,x) for (n,x) in enumerate(a)]
292:
293: function head!(i::Cols, n,txt)
294:   the = THE.char
295:   goalp() = the.less in txt || the.more in txt
296:   nump() = the.num in txt || goalp()
297:   yp() = klassp() || goalp()
298:   klassp() = the.klass in txt
299:   x = nump() ? Some : Sym
300:   y = x(pos=n, txt=txt)
301:   if klassp() i.klass = y end
302:   if goalp() push!(i.y.goals, y) end
303:   if nump()
304:     push!(i.nums,y); push!(yp() ? i.y.nums : i.x.nums, y)
305:   else
306:     push!(i.syms,y); push!(yp() ? i.y.syms : i.x.syms, y)
307:   end
308:   push!(yp() ? i.y.all : i.x.all, y)
309:   push!(i.all, y)
310: end
311:
312: #-----
313:
314: function tbl1(f="data/auto.csv")
315:   t = table(f)
316:   println("n ",length(t.rows))
317:   for col in t.cols.x.nums
318:     println(div(col)) #println(var(col)," ",col.all)
319:   end
320: end
321:
322: function nums(f="data/auto.csv")
323:   t = table(f)
324:   #println(t.rows[end].cells)
325:   for num in t.cols.x.nums
326:     d=div(num)
327:     println(num.txt, " ",length(d))
328:     println(d)
329:   end
330: end
331:
332: function syml()
333:   s=Sym()
334:   [add!(s,x) for x in "aaaabbc"]
335: end
336:
337: function Lines1(f="data/weather.csv")
338:   m=1
339:   print(m)
340:   for (n,tmp) in Lines(file= f)
341:     m += sizeof(tmp) #println(n," ",tmp)
342:     if mod(n,1000) == 0 println(n,":",m) end
343:   end
344:   print(m)
345: end
346:
347: function num1(x)
348:   if x<0.3 return 0.1 end
349:   if x<0.7 return 0.8 end
350:   return 0.9
351: end
352:
353: function numbers1(s=Some())
354:   [add!(s,num1(rand())) for i in 1:100]
355:   println([has(s,i) for i in div(s)])
356: end
357:
358: function numbers2(n=2, s=Some())
359:   [add!(s,rand()^0.5) for i in 1:10^n]
360:   println([ (i,has(s,i)) for i in div(s)])

```

```

361: end
362:
363: #somel()
364: #syml()
365: #@time tbl1("data/xomol0000.csv")
366: #@time tbl1("data/weather.csv")
367: #@time nums("data/xomol0000.csv")
368: #numbers1()
369: #@time numbers2(3)
370: end

```