```
#!/usr/bin/env lua
--
--             _ _ _   _   _        _ _ _
--    __ _   | (_) |_| |_| | ___  | (_) |_ ___
--   / _` |  | | | __| __| |/ _ \ | | | __/ _ \
--  | (_| |  | | | |_| |_| |  __/ | | | ||  __/
--   \__,_|  |_|_|\__|\__|_|\___| |_|_|\__\___|
--      __                 _                   _
--     / /    /\ /\   /_\  | | ___  __ _ _ __ _ __ (_)_ __   __ _
--    / /    / / \ \ //_\\ | |/ _ \/ _` | '__| '_ \| | '_ \ / _` |
--   / /___  \ \_/ //  _  \| |  __/ (_| | |  | | | | | | | | (_| |
--   \____/   \___/ \_/ \_/|_|\___|\__,_|_|  |_| |_|_|_| |_|\__, |
--                                                          |___/
--    _ _ _
--   | (_) |__  _ __ __ _ _ __ _   _
--   | | | '_ \| '__/ _` | '__| | | |
--   | | | |_) | | | (_| | |  | |_| |
--   |_|_|_.__/|_|  \__,_|_|   \__, |
--                             |___/
local the=require"z"{
what = "Small sample multi−objective optimizer.",
who  = "(c) 2021 Tim Menzies <timm@ieee.org> unlicense.org",
why  = [[
Sort N examples on multi−goals using a handful of 'hints'; i.e.

− Evaluate and rank, a few examples (on their y−values);
− Sort other examples by x−distance to the ranked ones;
− Recurse on the better half (so we sample more and more
  from the better half, then quarter, then eighth...).

A regression tree learner then explores the examples (sorted
left to right, worst to best).  By finding branches that
reduce the variance of the index of those examples, this
tree reports what attribute ranges select for the better (or
worse) examples.  ]],

how={{"FILE",      "−f",   "../../data/auto93.csv",   "read data from file"},
     {"CULL",      "−c",   .5     ,"cuts per generation"},
     {"HELP",      "−h",   false   ,"show help"                        },
     {"HINTS",     "−H",   4       ,"hints per generation"       },
     {"P",         "−p",   2       ,"distance calc exponent"     },
     {"SMALL",     "−s",   .5      ,"div list t into t^small"    },
     {"SEED",      "−S",   10019   ,"random number seed"           },
     {"TRAIN",     "−t",   .5      ,"size of training set"        },
     {"TODO",      "−T",   "all"   ,"run unit test, or 'all'"    },
     {"TRIVIAL",   "−v",   .35     ,"small delta=trivial*sd"     },
     {"WILD",      "−W",   false   ,"run tests, no protection"   }}}

local pop,push,shuffle,lap,last = the.pop,the.push,the.shuffle,the.lap,the.last
local fmt,out,shout             = the.fmt,the.out,the.shout
local sort,bchop                = the.sort, the.bchop
local obj,has                   = the.obj, the.has
local abs                       = the.abs
local csv                       = the.csv
--the==>it
--[[
Spans
 Little languages:
   − options
   − data language

Lesson plan
−− w1: ssytems: github. github workplaces. unit tests. doco tools.
−− w2: num,sym
−− W3: sample
−− w4: eval, knn, unfarinessness
−− W5:
−−]]
```

```
--      _        _ _       _      _ _      _  |   .   _
--     |‾|  |_|  |‾|‾|    _\ \/  |‾|‾|    _\ |<  |  |_)
--                             /                     |

−− ## Stuff for tracking `Num`bers.
−− `Num`s track a list of number, and can report  it sorted.
local Num=obj"Num"
function Num.new(inits,at, txt,     self)
  self= has(Num,{at=at or 0, txt=txt or"", w=(txt or""):find"−" and −1 or 1,
                 has={}, n=0, lo=1E32, hi =1E−32, ready=true})
  for _,one in pairs(inits or {}) do self:add(one) end
  return self end

function Num:add(x)
  if     x>self.hi then self.hi = x
  elseif x<self.lo then self.lo = x end
  push(self.has,x); self.n=self.n+1; self.ready=false end

−− Ensure that the returned list of numbers is sorted.
function Num:all(x)
  if not self.ready then table.sort(self.has) end
  self.ready = true
  return self.has end

function Num:dist(a,b)
  if     a=="?" then b=self:norm(b); a = b>.5 and 0 or 1
  elseif b=="?" then a=self:norm(a); b = a>.5 and 0 or 1
  else   a,b = self:norm(a), self:norm(b) end
  return abs(a−b) end

−− Combine two `num`s.
function Num:merge(other,    new)
  new = Num.new(self.has)
  for _,x in pairs(other.has) do new:add(x) end
  return new end

−− Return a merged item if that combination
−− is simpler than its parts.
function Num:mergeable(other,    new,b4)
  new = self:merge(other)
  b4  = (self.n*self:sd() + other.n*other:sd()) / new.n
  if b4 >= new:sd() then return new end end

−− The `mid` is the 50th percentile.
function Num:mid() return self:per(.5) end

−− Return `x` normalized 0..1, lo..hi.
function Num:norm(x,     lo,hi)
  if x=="?" then return x end
  lo,hi = self.lo, self.hi
  return abs(hi − lo) < 1E−32 and 0 or (x − lo)/(hi − lo) end

−− Return the `p`−th percentile number.
function Num:per(p,    t)
  t = self:all()
  p = p*#t//1
  return #t<2 and t[1] or t[p < 1 and 1 or p>#t and #t or p] end

−− The 10th to 90th percentile range is 2.56 times the standard deviation.
function Num:sd() return (self:per(.9) − self:per(.1))/ 2.56 end

−− Create one span holding  row indexes associated with each number
local div −− defined below
function Num:spans(egs)
  local xys,xs = {},  Num()
  for pos,eg in pairs(egs) do
    local x = eg[self.at]
    if x ~= "?" then
      xs:add(x)
      push(xys, {x=x,y=pos}) end end
  return div(xys,                  −− split xys into spans...
             xs.n^the.SMALL,        −− ..where spans are of size sqrt(#xs)..
             xs:sd()*the.TRIVIAL) end −− ..and spans have (last−first)>trivial

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−− ## Stuff for tracking `Sym`bol Counts.
−− `Sym`s track symbol counts and the `mode` (most frequent symbol).
local Sym=obj"Sym"
function Sym.new(inits,at,txt,     self)
  self= has(Sym,{at=at or 0, txt=txt or "", has={}, n=0, mode=nil, most=0})
  for _,one in pairs(inits or {}) do self:add(one) end
  return self end

function Sym:add(x)
  self.n = self.n + 1
  self.has[x] = 1 + (self.has[x] or 0)
  if self.has[x] > self.most then self.most, self.mode = self.has[x], x end end

function Sym:dist(a,b) return a==b and 0 or 1 end
function Sym:mid() return self.mode end

−− Create one span holding  row indexes associated with each symbol
function Sym:spans(egs,    xys,x)
   xys = {}
   for pos,eg in pairs(egs) do
     x = eg[self.at]
     if x ~= "?" then
       xys[x] = xys[x] or {}
       push(xys[x], pos)  end end
  return map(xys, function(x,t) return {lo=x, hi=x, has=Num(t)} end) end

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−− ## Stuff for skipping all things sent to a column
local Skip=obj"Skip"
function Skip.new(_,at,txt) return has(Skip,{at=at or 0, txt=txt or"", n=0}) end
function Skip:add(x) self.n = self.n + 1; return  x end
```

```lua
--      _   _   _ _    _    |   _
--     _\  (_|  | | |  |_)  |  (/_
--                     |

-- Samples store examples. Samples know about
-- (a) lo,hi ranges on the numerics
-- and (b) what  are independent `x` or dependent `y` columns.
local Sample = obj"Sample"
function Sample.new(     src,self)
  self = has(Sample,{names=nil, all={}, ys={}, xs={}, egs={}})
  if src then
    if type(src)=="string" then for x   in csv(src) do self:add(x)   end end
    if type(src)=="table" then for _,x in pairs(src) do self:add(x) end end end
  return self end

function Sample:add(eg,      ako,what,where)
  if not self.names
  then -- create the column headers
    self.names = eg
    for at,x in pairs(eg) do
      ako  = x:find":" and Skip or x:match"^[A-Z]" and Num or Sym
      what = push(self.all, ako({}, at, x))
      if not x:find":" then
        where = (x:find("+") or x:find("-")) and self.ys or self.xs
        push(where, what) end end
  else -- store another example; update column headers
    push(self.egs, eg)
    for at,x in pairs(eg) do if x ~= "?" then self.all[at]:add(x) end end end
  return self end

function Sample:better(eg1,eg2,      e,n,a,b,s1,s2)
  n,s1,s2,e = #self.ys, 0, 0, 2.71828
  for _,num in pairs(self.ys) do
    a  = num:norm(eg1[num.at])
    b  = num:norm(eg2[num.at])
    s1 = s1 - e^(num.w * (a-b)/n)
    s2 = s2 - e^(num.w * (b-a)/n) end
  return s1/n < s2/n end

function Sample:betters(egs)
  return sort(egs or self.egs,function(a,b) return self:better(a,b) end) end

function Sample:clone(      inits,out)
  out = Sample.new():add(self.names)
  for _,eg in pairs(inits or {}) do out:add(eg) end
  return out end

function Sample:dist(eg1,eg2,     a,b,d,n,inc)
  d,n = 0,0
  for _,col in pairs(self.xs) do
    a,b = eg1[col.at], eg2[col.at]
    inc = a=="?" and b=="?" and 1 or col:dist(a,b)
    d   = d + inc^the.P
    n   = n + 1 end
  return (d/n)^(1/the.P) end

-- Report mid of the columns
function Sample:mid(cols)
  return lap(cols or self.ys,function(col) return col:mid() end) end

-- Return spans of the column that most reduces variance
function Sample:splitter(cols)
  function worker(col) return self:splitter1(col) end
  return first(sort(lap(cols or sample.xs, worker), firsts))[2]  end

-- Return a column's spans, and the expected sd value of those spans.
function Sample:splitter1(col,     spans,xpect)
  spans= col:spans(self.egs)
  lap(spans,shout)
  --:xpect= sum(spans, function(_,span) return span.has.n*span.has:sd()/#self.egs end)
  return {xpect, spans} end

-- Split on column with best span, recurse on each split.
function Sample:tree(min,      node,min,sub,splitter, splitter1)
  node = {node=self, kids={}}
  min  = min  or (#self.egs)^the.SMALL
  if #self.egs >= 2*min then
    for _,span in pairs(self:splitter()) do
      sub = self:clone()
      for _,at in pairs(span.has) do sub:add(self.egs[at]) end
      push(node.kids, span)
      span.has = sub:tree(min) end end
  return node end

-- Find which leaf best matches an example `eg`.
function Sample:where(tree,eg,    max,x,default)
  if #kid.has==0 then return tree end
  max = 0
  for _,kid in pairs(tree.node) do
    if #kid.has > max then default,max = kid,#kid.has end
    x = eg[kid.at]
    if x ~= "?" then
      if x <= kid.hi and x >= kid.lo then
        return self:where(kid.has.eg) end end end
  return self:where(default, eg) end

------------------------------------------------------------------------------
-- Discretization tricks
-- Input a list of {{x,y}..} values. Return spans that divide the `x` values
-- to minimize variance on the `y` values.
function div(xys, tiny, dull,      merge)
  function merge(b4) -- merge adjacent spans if combo simpler to he parts
    local j, tmp = 0, {}
    while j < #b4 do
      j = j + 1
      local now, after = b4[j], b4[j+1]
      if after then
        local simpler = now.has:mergeable(after.has)
        if simpler then
          now = {lo=now.lo, hi= after.hi, has=simpler}
          j = j + 1 end end
      push(tmp,now) end
    return #tmp==#b4 and b4 or merge(tmp) -- recurse until nothing merged
  end --------------------
  local spans,span
  xys   = sort(xys, function(a,b) return a.x < b.x end)
  span  = {lo=xys[1].x, hi=xys[1].x, has=Num()}
  spans = {span}
  for j,xy in pairs(xys) do
    local x, y = xy.x, xy.y
    if    j < #xys - tiny   and    -- enough items remaining after split
          x ~= xys[j+1].x   and    -- next item is different (so can split here)
          span.has.n > tiny and    -- span has enough items
          span.hi - span.lo > dull -- span is not trivially small
    then span = push(spans, {lo=x, hi=x, has=Num()})  -- then new span
    end
    span.hi = x
    span.has:add(y) end
  return merge(spans) end

```

```lua
--    |_   .   _   _|_  .   _    _
--    | |  |  | |   |   |  | |  (_|
--                                _|

-- Sorting on a few y values
local hints={}
local copy,push,lap,sort=the.copy, the.push,the.lap,the.sort
local map,first,second,firsts=the.map, the.first, the.second, the.firsts

function hints.default(eg) return eg end

function hints.sort(sample,scorefun,    test,train,egs,scored,small)
  sample = Sample.new(the.FILE)
  train,test = {}, {}
  for i,eg in pairs(shuffle(sample.egs)) do
     push(i<= the.TRAIN*#sample.egs and train or test, eg) end
  egs = copy(train)
  small = (#egs)^the.SMALL
  local i=0
  scored = {}
  while #egs >= small do
    local tmp ={}
    i = i + 1
    io.stderr:write(fmt("%s",string.char(96+i)))
    for j=1,the.HINTS do
      egs[j] = (scorefun or hints.default)(egs[j])
      push(tmp, push(scored, egs[j]))
    end
    egs = hints.ranked(scored,egs,sample)
    for i=1,the.CULL*#egs//1 do pop(egs) end
  end
  io.stderr:write("\n")
  train=hints.ranked(scored, train, sample)
  return #scored, sample:clone(train), sample:clone(test) end

function hints.ranked(scored,egs,sample,worker,  some)
  function worker(eg) return {hints.rankOfClosest(scored,eg,sample),eg} end
  scored = sample:betters(scored)
  return  lap(sort(lap(egs, worker),firsts),second) end

function hints.rankOfClosest(scored,eg1,sample,        worker,closest)
  function worker(rank,eg2) return {sample:dist(eg1,eg2),rank} end
  closest = first(sort(map(scored, worker),firsts))
  return  closest[2] end --+ closest[1]/10^8 end

```

```lua
--  _|    _    _ _    _    _
-- (_|   (/_  | | |  (_)  _\

the.eg={}
local copy,rand =the.copy,the.rand
function the.eg.shuffle(   t)
  t={}
  for i=1,32 do push(t,i) end
  u = shuffle(copy(t))
  v = shuffle(copy(t))
  assert(#t == #u and u[1] ~= v[1]) end

function the.eg.lap()
  assert(3==lap({1,2},function(x) return x+1 end)[2]) end

function the.eg.map()
  assert(3==map({1,2},function(_,x) return x+1 end)[2]) end

function the.eg.tables()
  assert(20==sort(shuffle({{10,20},{30,40},{40,50}}),firsts)[1][2]) end

function the.eg.csv(   n,z)
  n=0
  for eg in the.csv(the.FILE) do n=n+1; z=eg end
  assert(n==399 and z[#z]==50) end

local first,rnds = the.first, the.rnds
function the.eg.rnds(    t)
  assert(10.2 == first(rnds({10.22,81.22,22.33},1))) end

function the.eg.sym(    s)
  s=Sym{"a","a","a","a","b","b","c"}
  assert("a"==s.mode) end

function the.eg.num1(    n)
  n=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  assert(.375 == n:norm(25))
  assert(15.625 == n:sd()) end

function the.eg.num2(    n1,n2,n3,n4)
  n1=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  n2=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  assert(n1:mergeable(n2)~=nil)
  n3=Num{10,20,30,40,50,10,20,30,40,50,10,20,30,40,50}
  n4=Num{100,200,300,400,500,100,200,300,400,500,100,200,300,400,500}
  assert(n3:mergeable(n4)==nil) end

function the.eg.sample(    s,tmp,d1,d2,n)
  s=Sample(the.FILE)
  assert(2110 == last(s.egs)[s.all[4].at])
  local sort1= s:betters(s.egs)
  local lo, hi = s:clone(), s:clone()
  for i=1,20                do lo:add(sort1[i]) end
  for i=#sort1,#sort1-30,-1 do hi:add(sort1[i]) end
  shout(s:mid())
  shout(lo:mid())
  shout(hi:mid())
  for m,eg in pairs(sort1) do
    n = bchop(sort1, eg,function(a,b) return s:better(a,b) end)
    assert(m-n <=2) end end

function the.eg.dists(    s,tmp,d1,d2,n)
  s=Sample(the.FILE)
  tmp = sort(lap(shuffle(s.egs),
                     function(eg2) return {s:dist(eg2,s.egs[1]), eg2} end),
               firsts)
   d1=s:dist(tmp[1][2], tmp[10][2])
   d2=s:dist(tmp[1][2], tmp[#tmp][2])
   assert(d1*10 < d2) end

function the.eg.binsym(   s,col)
  s=Sample(the.file)
  col = s.all[7]
  print(col.txt)
  s:splitter1(col)
  end

function the.eg.hints(    s,_,__,evals,sort1,train,test,n)
  s=Sample(the.FILE)
  evals, train,test = hints.sort(s)
  test.egs = test:betters()
  for m,eg in pairs(test.egs) do
    n = bchop(train.egs, eg,function(a,b) return s:better(a,b) end) end end

---| start-up |----------------------------------------------------------------------
the{demos=the.eg, nervous=true}

```

```lua
--[[
    _|_ _     _|  _
     | (_)  (_|(_)

the==>it

Spans
 Little languages:
   - options
   - data language

Lesson plan
- w1: ssytems: github. github workplaces. unit tests. doco tools.

- w2: num,sym
- W3: sample
- w4: eval, knn, unfarinessness
- W5:

-  seems to be  a revers that i  need to do .... but dont
- check if shuffle is working

teaching:
- sample is v.useful
--]]
```

```lua
--  _   _                               _   _        _      _
-- | |_(_)_ __  ___    __ _ _ __   __| | | |_ _ __(_) ___| | _____
-- | __| | '_ \/ __|  / _` | '_ \ / _` | | __| '__| |/ __| |/ / __|
-- | |_| | |_) \__ \ | (_| | | | | (_| | | |_| |  | | (__|   <\__ \
--  \__|_| .__/|___/  \__,_|_| |_|\__,_|  \__|_|  |_|\___|_|\_\___/
--       |_|

local lib={}

--    _  _   _        _  _
--  |  (_) (_| |_| (/__\
--           _|
lib._b4={}; for k,v in pairs(_ENV) do lib._b4[k]=k end
function lib.rogues()
  for k,v in pairs(_ENV) do
    if not lib._b4[k] then print("?rogue: ",k,type(v)) end end end

--    _  _  _   _|  _  _ _  _
--  |  (_|| | (_| (_) | | |_\
--
lib.Seed = 10019
-- random integers
function lib.randi(lo,hi) return math.floor(0.5 + lib.rand(lo,hi)) end
-- random floats
function lib.rand(lo,hi,     mult,mod)
  lo, hi = lo or 0, hi or 1
  lib.Seed = (16807 * lib.Seed) % 2147483647
  return lo + (hi-lo) * lib.Seed / 2147483647 end

-- _|_  _ |_  |  _
--  |  (_||_) | (/_
--
-- Table to string.
lib.cat     = table.concat
-- Return a sorted table.
lib.sort    = function(t,f) table.sort(t,f); return t end
-- Return first,second, last  item.
lib.first   = function(t) return t[1] end
lib.second  = function(t) return t[2] end
lib.last    = function(t) return t[#t] end
-- Function for sorting pairs of items.
lib.firsts  = function(a,b) return a[1] < b[1] end
-- Add to end, pull from end.
lib.pop     = table.remove
lib.push    = function(t,x) table.insert(t,x); return x end

-- Random order of items in a list (sort in place).
function lib.shuffle(t,   j)
  for i=#t,2,-1 do j=lib.randi(1,i); t[i],t[j]=t[j],t[i] end; return t end

-- Collect values, passed through 'f'.
function lib.lap(t,f)  return lib.map(t,f,1) end
-- Collect key,values, passed through 'f'.
-- If `f` returns two values, store as key,value.
-- If `f` returns one values, store at index value.
-- If `f' return nil then add nothing (so `map` is also `select`).
function lib.map(t,f,one,     u)
  u={}; for x,y in pairs(t) do
    if one then x,y=f(y) else x,y=f(x,y) end
    if x ~= nil then
      if y then u[x]=y else u[1+#u]=x end end end
  return u end

-- Shallow copy
function lib.copy(t,  u) u={}; for k,v in pairs(t) do u[k]=v end; return u end

function lib.top(t,n,      u)
  u={};for k,v in pairs(t) do if k>n then break end; push(u,v) end; return u;end

--- Return a table's keys (sorted).
function lib.keys(t,u)
  u={}
  for k,_ in pairs(t) do if tostring(k):sub(1,1)~="_" then lib.push(u,k) end end
  return lib.sort(u) end

-- Binary chop (assumes sorted lists)
function lib.bchop(t,val,lt,lo,hi,     mid)
  lt = lt or function(x,y) return x < y end
  lo,hi = lo or 1, hi or #t
  while lo <= hi do
    mid =(lo+hi) // 2
    if lt(t[mid],val) then lo=mid+1 else hi= mid-1 end end
  return math.min(lo,#t)  end

--   _  _   _ _|_|_  _
--  | | | |(_| |  | |_\
--
lib.abs = math.abs
-- Round `x` to `d` decimal places.
function lib.rnd(x,d,  n) n=10^(d or 0); return math.floor(x*n+0.5) / n end
-- Round list of items to  `d` decimal places.
function lib.rnds(t,d)
  return lib.lap(t, function(x) return lib.rnd(x,d or 2) end) end

-- Sum items, filtered through `f`.
function lib.sum(t,f)
  f= f or function(x) return x end
  out=0; for _,x in pairs(f) do out = out + f(x) end; return out end

--   _  _ . _ _|_ . _  _
--  |_)|  || | |  || |(_|
--  |                  _|
lib.fmt = string.format
lib.say = function(...) print(lib.fmt(...)) end

-- Print as red, green, yellow, blue.
function lib.color(s,n) return lib.fmt("\27[1m\27[%sm%s\27[0m",n,s) end
function lib.red(s)     return lib.color(s,31) end
function lib.green(s)   return lib.color(s,32) end
function lib.yellow(s)  return lib.color(s,34) end
function lib.blue(s)    return lib.color(s,36) end

-- Printed string from a nested structure.
lib.shout = function(x) print(lib.out(x)) end
-- Generate string from a nested structures
-- (and don't print any contents more than once).
function lib.out(t,seen,    u,key,value,public)
  function key(k) return lib.fmt(":%s %s", lib.blue(k), lib.out(t[k],seen)) end
  function value(v) return lib.out(v,seen) end
  if type(t) == "function" then return "(...)" end
  if type(t) ~= "table"    then return tostring(t) end
  seen = seen or {}
  if seen[t] then return "..." else seen[t] = t end
  u = #t>0 and lib.lap(t, value) or lib.lap(lib.keys(t), key)
  return lib.red((t._is or"").."{")..lib.cat(u," ")..lib.red("}") end

--   |`.|  _  _
--  ~|~|| (/__\
--
-- Return one table per line, split on commas.
function lib.csv(file,   line)
  file = io.input(file)
  line = io.read()
  return function(   t,tmp)
    if line then
      t={}
      for cell in line:gsub("[\t\r ]*",""):gsub("#.*",""):gmatch("([^,]+)") do
        lib.push(t, tonumber(cell) or cell) end
      line = io.read()
      if #t>0 then return t end
    else io.close(file) end end end

```

```lua
--   _ |_ . _  _ _|_ _
--  (_)|_)|(/_(_  |  _\
--       L|
-- Create an instance
function lib.has(mt,x) return setmetatable(x,mt) end
-- Create a clss
function lib.obj(s, o,new)
   o = {_is=s, __tostring=out}
   o.__index = o
   return setmetatable(o,{__call = function(_,...) return o.new(...) end}) end

--   _  _  _ _  _ _  _  _  _|   |. _  _
--  (_ (_)| | || | || |(_|| |(_|   ||| |(/_
--
function lib.help(about)
  lib.say("\n%s [OPTIONS]\n%s\n%s\n\nOPTIONS:\n",
            arg[0], about.who, about.what)
  for _,t in pairs(about.how) do
    lib.say("%4s %-9s%-30s%s %s",
                t[2],t[3] and t[1] or"", t[4],t[3] and"=" or"",t[3] or"") end
  print("\n"..about.why) end

function lib.cli(about,u)
  u={}
  for _,t in pairs(about.how) do -- update defaults from command line
    u[t[1]] = t[3]
    for n,word in ipairs(arg) do if word==t[2] then
      local new = t[3] and (tonumber(arg[n+1]) or arg[n+1]) or true
      assert(type(new) == type(u[t[1]]), word.." expects a "..type(u[t[1]]))
      u[t[1]] = new end end end
  lib.Seed = u.seed or 10019
  if u.HELP then lib.help(about); os.exit() end
  return u end

--    __|_ _  __|_   |_ |_
--   _\ | (_||  |    |_||_)
--                      |
-- make everything  the. the.Eg,
-- assumes the, about, eg
function lib.theMain(settings,demos,    defaults,fails)
  defaults={}
  for k,v in pairs(settings) do defaults[k]=v end
  fails=0
  local function example(k,      f,ok,msg)
    f= demos[k]
    assert(f,"unknown action "..k)
    for k,v in pairs(defaults) do settings[k]=v end
    lib.Seed  = settings.SEED or 10019
    if settings.WILD then return f() end
    ok,msg = pcall(f)
    if ok then print(lib.green("PASS"),k)
    else       print(lib.red("FAIL"),  k,msg); fails=fails+1 end
  end ---------------------
  if     settings.TODO == "all"
  then   settings.lap(lib.keys(demos),example)
  elseif settings.TODO == "ls"
  then   print("\nACTIONS:")
         lib.map(lib.keys(demos),function(_,k) print("\t"..k) end)
  else   example(settings.TODO)
  end
  lib.rogues()
  return os.exit(fails) end

--    _  _ _|__|_. _  _    _
--   _\(/_ |   | || |(_|  |_||_)
--                    _|     |
-- return all the above functions, augmented with
-- (1) any update on the constants from the command line;
-- (2) a call method that offer some extra services.
-- To avoid name classes (of config settings and functions),
-- always use UPPER CASE for the variables and lower case for
-- the first letter of the functions.
return function(t)
  local function main(settings,actions)
    for flag,val in pairs(actions or {}) do
      if flag=="nervous" and val then lib.rogues() end
      if flag=="demos"           then lib.theMain(settings,val) end end
    return t end
  t=lib.cli(t)
  for k,v in pairs(lib) do t[k] = v end
  return setmetatable(t, {__call=main}) end
```