```lua
local l={}

-- ----------------------------------------------------------------------------
-- ## Lint

-- To find rogue variables, call `rogues()` very last thing.
local b4={}; for k,v in pairs(_ENV) do b4[k]=v end
function l.rogues()
  for k,v in pairs(_ENV) do if not b4[k] then print("?",k,type(v)) end end end

-- ----------------------------------------------------------------------------
-- ## Lists

-- Deep copy
function l.copy(t1,     t2)
  if type(t1) ~= "table" then return t1 end
  t2={}; for k,v in pairs(t1) do t2[l.copy(k)] = l.copy(v) end
  return setmetatable(t2,getmetatable(t1))  end

-- Return the `n`-th thing from the sorted list `t`. e.g median is per(t.5).
function l.per(t,n)
  n=math.floor(((n or .5)*#t)+.5); return t[math.max(1,math.min(#t,n))] end

-- Add to `t`, return `x`.
function l.push(t,x) t[1+#t]=x; return x end

-- Sort, return sorted list
function l.sort(t,fun) table.sort(t,fun); return t end

-- ----------------------------------------------------------------------------
-- ### Maths

-- Round `n` to `nPlaces`.
function l.rnd(n, nPlaces,     mult)
  mult = 10^(nPlaces or 2)
  return math.floor(n * mult + 0.5) / mult end

-- ----------------------------------------------------------------------------
-- ### Stings
--
-- Convert string `s` to boolean, int, float or failing all else, string.
function l.coerce(s,     fun)
  function fun(s1)
    if s1=="true"  then return true end
    if s1=="false" then return false end
    return s1 end
  return math.tointeger(s) or tonumber(s) or fun(s:match"^%s*(.-)%s*$") end

-- `o` is a telescopt and `oo` are some binoculars we use to exam stucts.
-- `o`:  generates a string from a nested table.
function l.o(t,    show,u)
  if type(t) ~=  "table" then return tostring(t) end
  function show(k,v)
    if not tostring(k):find"^_"  then
      v = l.o(v)
      return #t==0 and string.format(":%s %s",k,v) or tostring(v) end end
  u={}; for k,v in pairs(t) do u[1+#u] = show(k,v) end
  if #t==0 then table.sort(u) end
  return "{"..table.concat(u," ").."}" end

-- `oo`: prints the string from `o`.
function l.oo(t) print(l.o(t)) return t end

-- ----------------------------------------------------------------------------
-- ## Files
--
-- Call `fun` on each row. Row cells are divided in `the.seperator`.
function l.csv(fname,fun,      src,s,t)
  src = io.input(fname)
  while true do
    s = io.read()
    if not s then return io.close(src) else
      t={}
      for s1 in s:gmatch("([^,]+)") do t[1+#t] = l.coerce(s1) end
      fun(t) end end end

-- ----------------------------------------------------------------------------
-- ## Settings

-- Parse `s` for lines containing options (newline, space, dash)
function l.settings(s,    t)
  t={_help = s}
  s:gsub("\n [-][%S]+[%s]+[-][-]([%S]+)[^\n]+= ([%S]+)",
          function(k,x) t[k] = l.coerce(x) end)
  return t end

-- Update settings from values on command-line flags.
-- Booleans need no values (we just flip the defaults).
function l.cli(t)
  for slot,v in pairs(t) do
    v = tostring(v)
    for n,x in ipairs(arg) do
      if x=="-"..(slot:sub(1,1)) or x=="--"..slot then
        v = v=="false" and "true" or v=="true" and "false" or arg[n+1] end end
    t[slot] = l.coerce(v) end
  if t.help then os.exit(print("\n"..help.."\n")) end
  return t end

-- ----------------------------------------------------------------------------
-- ## Demos

-- Run1 demo
-- 1. reset random number seed before running something.
-- 2. Cache the detaults settings, and...
-- 3. ... restore them after the test
-- 4. Print error messages or stack dumps as required.
-- 5. Return true if this all went well.
function _run1(k,settings,funs,     old,status,out,msg)
  if not funs[k] then return end
  math.randomseed(settings.seed) -- reset seed [1]
  old={}; for k,v in pairs(settings) do old[k]=v end --  [2]
  if settings.dump then -- [4]
    status,out=true, funs[k]()
  else
    status,out=pcall(funs[k]) -- pcall means we do not crash and dump on errror
  end
  for k,v in pairs(old) do settings[k]=v end -- restore old settings [3]
  msg = status and ((out==true and "PASS") or "FAIL") or "CRASH" -- [4]
  print("!!!!!!", msg, k, status)
  return out and status end

-- Run demo `k` (which is an index of `funs`),
-- or if k==ls then list demo names,
-- or if k==all the run all demos.
-- Return to operating the number of failures.
function runs(k,settings,funs,     names,fails)
  settings = cli(settings)
  fails=0
  names={}; for s,_ in pairs(eg) do push(names,s) end
  if k == "ls" then
    print("\nExamples -e ...")
    for _,s in pairs(sort(names)) do print(string.format("\t%s",s))  end
  elseif k == "all" then
    for _,s in pairs(sort(names)) do
      print"\n----------------------------------"
      if not _run1(s,settings,funs) then fails=fails+ 1 end end
      rogues()
  else
    if not _runs(k,settings,funs) then fails=fails+1 end
  end
  rogues()
  os.exit(fails) end

-- ----------------------------------------------------------------------------
-- ### Objects
-- Constructor.
local function _new(klass,...)
  local inst=setmetatable({},klass);
  return setmetatable(klass.new(inst,...) or inst,klass) end

-- obj("Thing") enables a constructor Thing:new() ... and a pretty-printer
-- for Things.
function l.obj(s,    t)
  t={__tostring = function(x) return s..o(x) end}
  t.__index = t;return setmetatable(t,{__call=_new}) end

-- ----------------------------------------------------------------------------
-- That's all folks.
return l
```