```lua
local _ = {}

-- ## Maths Tricks

-- **r()**:  Random number shorthand.
_.r=math.random

-- **ish()**: is `x` is close-ish to `y`?
-- **cosine()**: for three  ABC with sides abc,
-- where does C falls on the line running AB?
function _.ish(x,y,z)  return math.abs(y -x ) < z end
function _.cosine(a,b,c)
  return math.max(0,math.min(1, (a^2+c^2-b^2)/(2*c+1E-32))) end

-- ## List Tricks

-- **any()**: returns any thing from a list
-- **many()**: return multiple **any()** things.
function _.any(a)        return a[ math.random(#a) ] end
function _.many(a,n,  u) u={}; for j=1,n do u[1+#u] =_.any(a) end; return u end

-- **last()**: last item in a list
-- ##per()**: p-th item in a list
function _.last(a)       return a[ #a ] end
function _.per(a,p)      return a[ (p*#a)//1 ] end

-- **pop()**: dump from end
-- **push()**: add to ed
function _.pop(a)        return table.remove(a) end
function _.push(t,x)     t[1 + #t] = x; return x end

-- **sort()**: return a list, ordered on function `f`.
-- **firsts()**:  order on sub-list first items
function _.sort(t,f)     table.sort(t,f); return t end
function _.firsts(a,b)   return a[1] < b[1] end

-- **map()**: return a list with `f` run over all items
function _.map(t,f, u) u={};for k,v in pairs(t) do u[1+#u]=f(v) end;return u end

-- **sum()**: sum all list items, filtered through `f`
-- (which defaults to just use the ran values).
function _.sum(t,f, n)
  n=0; _.map(t,function(v) n=n+(f and f(v) or v) end)
  return n end

-- **inc()** incretements a 1,2, or 3 nested dictionary counter
function _.inc(f,a,n)       f=f or{}; f[a]=(f[a] or 0) + (n or 1);   return f end
function _.inc2(f,a,b,n)    f=f or{}; f[a]=_.inc( f[a] or {},b,n);   return f end
function _.inc3(f,a,b,c,n) f=f or{}; f[a]=_.inc2(f[a] or {},b,c,n); return f end

-- **has()** implements a 1,2, or level nested lookup
function _.has(f,a)       return f[a]                  or 0 end
function _.has1(f,a,b)    return f[a] and _.has( f[a],b)   or 0 end
function _.has2(f,a,b,c) return f[a] and _.has1(f[a],b,c) or 0 end


-- **shuffle()**: randomize order (sorts in  place)
function _.shuffle(t,   j)
  for i=#t,2,-1 do j=math.random(i); t[i],t[j]=t[j],t[i] end; return t end

-- ## String -> Things

-- **words()**: split  string into list of substrings
function _.words(s,sep,   t)
  sep="([^" .. (sep or ",")  .. "]+)"
  t={}; for y in s:gmatch(sep) do t[1+#t] = y end; return t end

-- **things()**: convert strings in a list to things
-- **thing()**: convert string to a thing
function _.things(s) return _.map(_.words(s), _.thing) end
function _.thing(x)
  x = x:match"^%s*(.-)%s*$"
  if x=="true" then return true elseif x=="false" then return false end
  return tonumber(x) or x end

-- **lines()**: (iterator) return lines in a file. Standard usage is
-- `for cells in file(NAME,things) do ... end`
function _.lines(file,f,      x)
  file = io.input(file)
  f    = f or _.things
  return function() x=io.read(); if x then return f(x) else io.close(file) end e
nd end

-- ## Things -> Strings

-- **fmt()**:  String format shorthand
_.fmt = string.format

-- **oo()**: Print string from nested table.
-- **o()**: Generate string from nested table.
function _.oo(t) print(_.o(t)) end
function _.o(t,  seen, u)
  if type(t)~="table" then return tostring(t) end
  seen = seen or {}
  if seen[t] then return "..." end
  seen[t] = t
  local function show1(x) return _.o(x, seen) end
  local function show2(k) return _.fmt(":%s %s",k, _.o(t[k],seen)) end
  u = #t>0 and _.map(t,show1) or _.map(_.slots(t),show2)
  return (t._is or "").."{"..table.concat(u," ").."}" end

-- **slots()**: return table slots, sorted.
function _.slots(t, u)
  local function public(k) return tostring(k):sub(1,1) ~= "_" end
  u={};for k,v in pairs(t) do if public(k) then u[1+#u]=k end end
  return _.sort(u) end

-- **rnds()**: round list of numbers
-- **rnd()**: round one number.
function _.rnds(t,f) return map(t, function(x) return _rnd(x,f) end) end
function _.rnd(x,f)
  f = not f and "%s" or number and fmt("%%%sf",f) or f
  return fmt(type(x)=="number" and (x~=x//1 and f) or "%s",x) end

-- ## Make settings from help string  and CLI (command-line interface)

-- **cli()**: In a string, look for lines indented with two spaces, starting wit
h a dash.
-- Each such  line should have  a long and short flag, some help tesx
-- and (at end of line), a  default values. e.g.
--
--     -seed -S set the random number seed  = 10019
--
-- Each line generates  a setting  with key "seed" and
-- default value "10019". If the command line contains one of the flags
-- (`-seed` or `-s`) then update those defaults.
function _.cli(help,    d)
  d={}
  help:gsub("\n  ([-]([^%s]+))[%s]+([-][^%s]+)[^\n]*%s([^%s]+)",
    function(long,key,short,x)
      for n,flag in ipairs(arg) do
        if flag==short or flag==long then
          x = x=="false" and true or x=="true" and "false" or arg[n+1] end end
      d[key] = x==true and true or _.thing(x) end)
  if d.help then os.exit(print(help)) end
  return d end

-- ## Test suites

-- **ok()**: maybe, print stack dump on errors.
-- Increment the `fails` counter on failed `test`.
function _.ok(tests,test,msg)
  print(test and "  PASS: "or "  FAIL: ",msg or "")
  if not test then
    tests._fails = tests._fails+1
    if the and the.dump then assert(test,msg) end end end

-- **go()**:  run some `tests`, controlled by `settings`.
-- Maybe update the `_fails` counter.
-- Return the total fails to the operating system.
function _.go(tests,b4)
  tests._fails = 0
  local todo = the and the.todo or "all"
  for k,one in pairs(todo=="all" and _.slots(tests) or {todo}) do
    if k ~= "main" and type(tests[one]) == "function" then
      math.randomseed(the and the.seed  or 1)
      print(_.fmt("#%s",one))
      tests[one](tests) end end
  if b4 then
    for k,v in pairs(_ENV) do
      if not b4[k] then print("??",k,type(v)) end end end
  os.exit(tests._fails) end

-- ## Objects

-- **new()**:  make a new instance.
-- **class()**: define a new class of instances
_.new = setmetatable
function _.class(s,   t)
  t={__tostring=_.o,_is=s or ""}; t.__index=t
  return _.new(t, {__call=function(_,...) return t.new(_,...) end}) end

-- ## Return
return _
```