**july.jl**

```julia
module July
  help = """
         JULY : fun stuff
         (c)2022 Tim Menzies, timm@ieee.org, BSD-2

         USAGE: julia july.jl [OPTIONS]

         OPTIONS:
           -h  --help  show help            = false
           -p  --p     distance coeffecient = 2
           -s  --seed  random number seed   = 10019
         """
  using Random, Parameters
  includes(dir,files) = map(f->include("../src/$dir/$f.jl"),files)
  includes("lib", ["2thing","settings","2string","lists"])
  includes("col", ["col" ,"sample"])
  the = cli(settings(help))
end
```

**col/col.jl**

```julia
"Add stuff to `i`. Ignore unknown values. Increment `n`, call `inc1!`."
function inc!(i,a::Array) for x in a inc!(i,x,1) end;  a end
function inc!(i,x, n=1)
  if x != the[:unknown] i.n = i.n + n
                        inc1!(i,x,n) end  end
```

**col/sample.jl**

```julia
"Keep, at most `the[:max]` items."
@with_kw mutable struct Sample
  _has=[]      # where we keep, at most, the[:sample] items
  ok=false end # true if we have sorted the _has since last addition

"Add something to `_has`. If full, replace anything at random."
function inc1!(i::Sample,x,n) # <== tedious detail, ignore n (used only in Sym)
  n = length(i._has)
  if     ( n      < the[:max] ) begin i.ok=false; push!(i._has,x)  end
  elseif ( rand() < n/i.n )     begin i.ok=false; i._has[int(n*rand())+1]=x end end end

" `mid` = median. `div` = standard deviation. `per` returns the n-th item."
mid(i::Sample,     a=nums(i)) = per(a,.5)
div(i::Sample,     a=nums(i)) = (per(a,.9) - per(a, .1)) / 2,58
nums(i::Sample) = begin ( !i.ok || sort!(i._has) ) ; i.ok=true ; i._has end
```

**lib/2string.jl**

```julia
"print a struct"
function say(i)
  s,pre="$(typeof(i)){",""
  for f in sort!([x for x in fieldnames(typeof(i)) if !("$x"[1] == '_')])
    s,pre = s * pre * ":$f $getfield(i,f)" ," " end
  print(s * "}") end
```

**lib/2thing.jl**

```julia
"Coerce string to thing."
function coerce(s)
  for t in [Int64,Float64,Bool] if (x=tryparse(t,s)) != nothing return x end end
  return strip(s) end

"Coerce csv rows to cells."
function csv(file, fun)
  for line in eachline(file)
    line = strip(line)
    if sizeof(line) > 0 fun(map(coerce, split(line, ","))) end end end
```

**lib/lists.jl**

```julia
"Return the n-th item of `a`. e.g. `per(a,.5)` returns median."
per(a, n) = begin l=length(a); a[max(1,min(l,1 + trunc(Int,n*l)))] end
```

**lib/settings.jl**

```julia
"For e.g. slot x=1, update if cli has `-x 10`. For bool, cli flags flip default."
function cli(d::Dict)
  for (slot, x) in d
    for (i, v) in pairs(ARGS)
      if v == "-" * "$slot"[1]
        d[slot] = coerce(x==true  ? "false" : (
                         x==false ? "true"  : (
                         ARGS[i+1]))) end end end; d end

"Return dictonary of settings, extracted from help string."
function settings(s) # -> dictionary of settings
  d=Dict()
  # for example:        -h --help  show help = false
  for m in eachmatch(r"\n\s+-[^-]+--(\S+)[^=]+=\s+(\S+)",s)
    d[Symbol(m[1])] = coerce(m[2] ) end; d end
```