### User-built Functions/Procedures: load

- Two ways to **load** existing files containing functions/proc
  - `load "/path/my_script.ncl"`
  - use environment variable: `NCL_DEF_SCRIPTS_DIR`

- Similar to (say) python: **import**

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/diagnostics_cam.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/shea_util.ncl"
```
Examine contents scripts distributed with NCL

```
less $NCARG_ROOT/lib/ncarg/nclscripts/csm/diagnostics_cam.ncl
less $NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl
```

Use any editor to extract code and modify for you needs

```
vi $NCARG_ROOT/lib/ncarg/nclscripts/csm/diagnostics_cam.ncl
```
User built Functions/Procedures: **Scope**

- User functions must be **loaded** **prior** to use
  - unlike in compiled language (fortran C, C++)
- Built-in functions are **always** available

```ncl
load "dummy_1.ncl" ; not aware of constants/scripts below
gravity = 9.8
rgas = 204
load "dummy_2.ncl" ; can use gravity, rgas, dummy_1
rearth = 6371.009
load "dummy_3.ncl" ; can use gravity, rgas, rearth, _1 & _2

begin ; MAIN: can use all of the above
  ...
end
```
User-Built Functions/Procedures: **Purpose**

<table>
<thead>
<tr>
<th>function</th>
<th>returns one or more variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedure</td>
<td>perform a task (eg: create a plot / file)</td>
</tr>
</tbody>
</table>

**Feature:** Automatic ‘garbage’ collection. No need to explicitly delete variables prior to returning. NCL does it for you.
User-built Functions/Procedures: Sample

**myLib.ncl**

```ncl
undef ("mult")
function mult(x1,x2,x3,x4)
local sx1, foo
begin
    sx1 = sin(0.01745329*x1)
    foo = sx1*x2*x3*sqrt(x4)
    foo@long_name = "result"
    foo@units = "???
return (foo)
end
end

load "/some/path/myLIB.ncl"

begin
    x = mult(4.7, 34, 567, 2)
    print(x)
end
```

**NOTE:** `myLib.ncl` can contain multiple scripts
User-Built Functions/Procedures: Structure

- Development process similar to Fortran/C/IDL/Matlab

General Structure

```plaintext
#define ("function_name") ; optional
function function_name (declaration_list)
local local_identifier_list ; optional
begin ; required
 ... statements ...
   return (return_value) ; required
end ; required
```

```plaintext
#define ("procedure_name") ; optional
procedure procedure_name (declaration_list)
local local_identifier_list ; optional
begin ; required
 ... statements ...
end ; required
```
User-Built Functions/Procedures: Arguments

- arguments are passed by reference [fortran]
- argument **prototyping** (*optional*)
  - built-in functions are prototyped
- no type, no dimension specification
  - **procedure** whatever (a, b, c)
- constrained argument specification
  - require specific type, dimensions, and size
  - **procedure** ex(data[*][*]:float,res:logical,text:string)
- generic specification
  - type only
  - **function** xy_interp(x1:numeric, x2:numeric)
- combination
  - **function** ex (d[*]:float,x:numeric,wks:graphic,y[2], a)
User-Built Functions/Procedures: Optional Arg

- additional (‘optional’) arguments possible
- attributes associated with one or more arguments
  - often implemented as a separate argument (not required)

```plaintext
optArg = True
optArg@scale = 0.01
optArg@add = 1000
optArg@wgts = (/1,2,1/)
optArg@name = "sample"
optArg@array = array_3D

ex(x2D, "Example", optArg)

procedure ex(data, text, opt:logical)
begin
  if (opt .and. isatt(opt,"scale")) then
    d = data*opt@scale
  end if
  if (opt .and. isatt(opt,"wgts")) then
    : 
  end if
  if (opt .and. isatt(opt,"array")) then
    xloc3D = opt@array_3D ; nD arrays
  end if
  ; must be local before use
end
```
Command Line Arguments [CLAs]

- CLAs are NCL statements on the command line
  http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclCLO.shtml

```ncl
tStrt=1930 'lev=(/250, 750/)' 'var="T"' 'fNam="foo.nc"' sample.ncl
```

```ncl
if (.not. isvar("fNam") .and. (.not. isvar("var")) ) then
  print("fNam and/or variable not specified: exit")
  exit
end if

f = addfile (fNam, "r") ; read file
x =f->$var$ ; read variable

if (.not. isvar("tStrt")) then ; CLA?
  tStrt = 1900 ; default
end if

if (.not. isvar("lev")) then ; CLA?
  lev = 500 ; default
end if
```
View Sample User Written Functions

- **http://www.cgd.ucar.edu/~shea/meteo.ncl**
  - Pot. Temp; Static Stability; Pot. Vorticity (hybrid, isobaric)
  - Advect Variable (q): $u^* (dq/dx + v* dq/dy)$

- **http://www.cgd.ucar.edu/~shea/reg_func.ncl**
  - Multiple Linear regression: ANOVA
  - Simple Linear Regression: ANOVA
User-built Functions/Procedures: Return n Variables

```plaintext
undef("static_stability_n")

function static_stability_n (p:numeric, t:numeric, npr[1]:integer, sopt:integer)
[snip]
;     sopt    - =0, Return static stability only
;                - =1, Return static stability, theta, dthdp as type list
[snip]
begin
[snip]

if (sopt.eq.0) then
    return(s)
else
    dthdp@long_name = "vertical derivative of theta with pressure"
    dthdp@units     = s@units
    copy_VarCoords(t,dthdp)
    return( [/ s, theta, dthdp ] )
end if

end
```
User-built Functions/Procedures: Access List Var

load "./meteo.ncl"

... statements...

sopt = 0
s = static_stability_n (p, t, ndim, sopt)
......
sopt = 1
ss = static_stability_n (p, t, ndim, sopt)

s = ss[0]; extract static stability
theta = ss[1]; theta
dtdp = ss[2]; vertical derivative

delete( ss ); no longer needed; delete is not required

NOTE: No need to create new variables from the list.

ss[0], ss[1], ss[2] could be used directly
However, code clarity is improved by extraction.