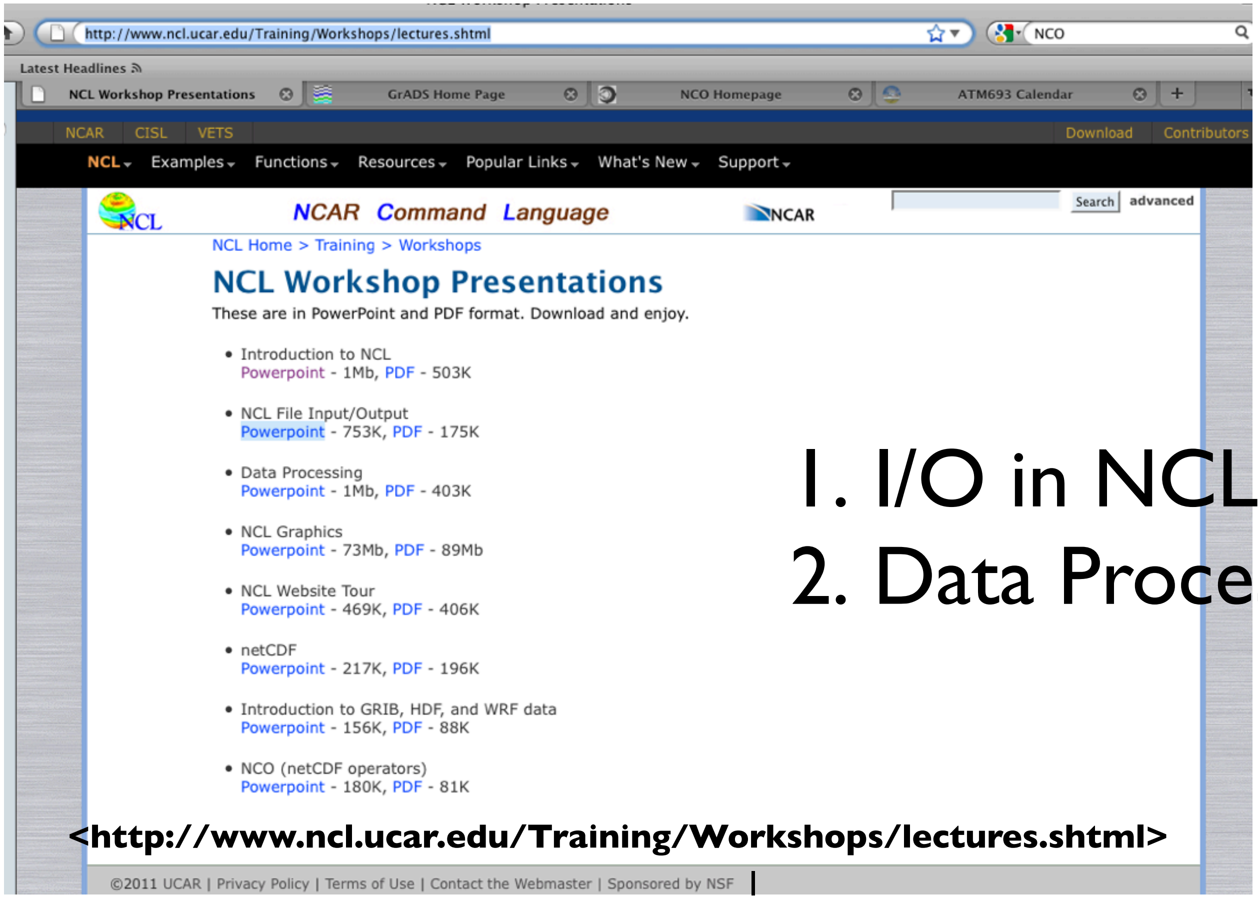
Class #4 Monday 31 January 2011

- 9:45-10:45, Schedule next week. No travel until late March
- What did we discuss last time?
- Today (1.3.3 NCl Part 1 and 2)
- Continuing with NCL tutorials



NCL Home > Training > Workshops

NCL Workshop Presentations

These are in PowerPoint and PDF format. Download and enjoy.

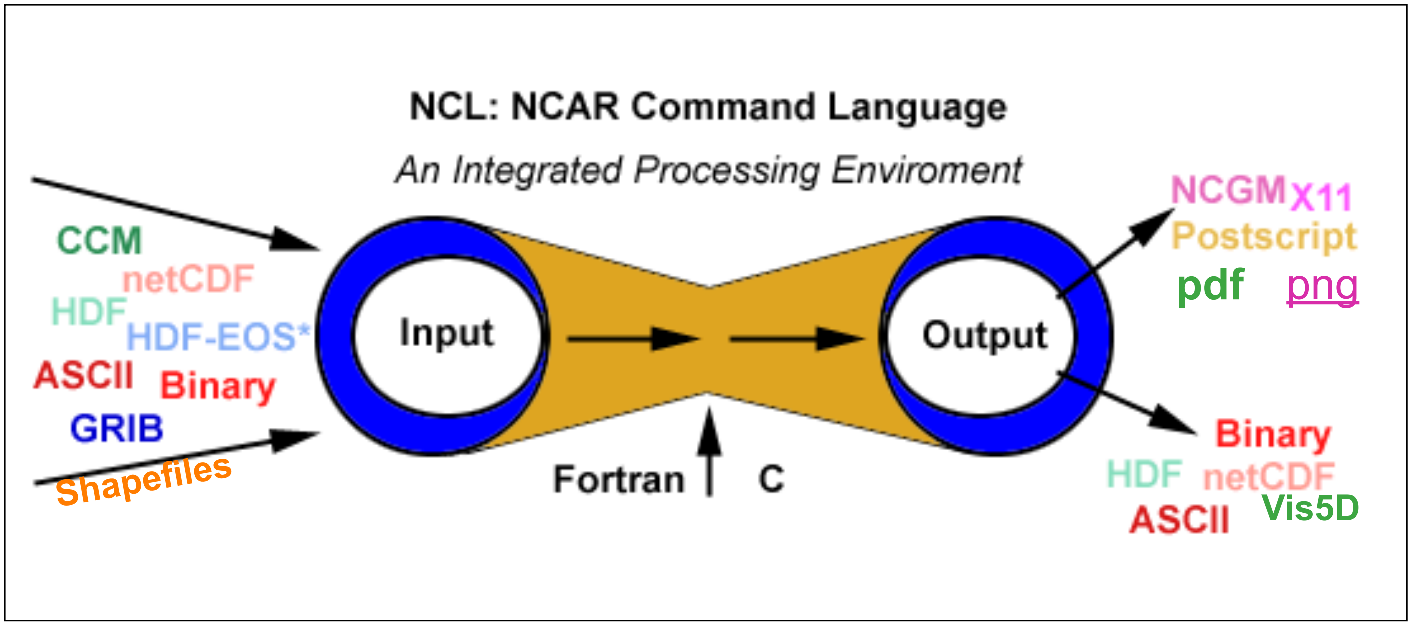
- Introduction to NCL
  Powerpoint - 1Mb, PDF - 503K
- NCL File Input/Output
  Powerpoint - 753K, PDF - 175K
- Data Processing
  Powerpoint - 1Mb, PDF - 403K
- NCL Graphics
  Powerpoint - 73Mb, PDF - 89Mb
- NCL Website Tour
  Powerpoint - 469K, PDF - 406K
- netCDF
  Powerpoint - 217K, PDF - 196K
- Introduction to GRIB, HDF, and WRF data
  Powerpoint - 156K, PDF - 88K
- NCO (netCDF operators)
  Powerpoint - 180K, PDF - 81K

©2011 UCAR | Privacy Policy | Terms of Use | Contact the Webmaster | Sponsored by NSF

1. I/O in NCL
2. Data Processing

<http://www.ncl.ucar.edu/Training/Workshops/lectures.shtml>

NCL File IO



Dennis Shea

National Center for Atmospheric Research

NCAR is sponsored by the National Science Foundation

setfileoption

www.ncl.ucar.edu/Document/Functions/Built_in/setfileoption.shtml

- **allows user to specify file-format-specific options**
  - netCDF, GRIB and Binary options *[currently]*
- **sample usage of selected options**

documentation

  - writing netCDF
    - **setfileoption**(f, "DefineMode" ,True)
  - reading GRIB
    - **setfileoption**("grb" ,"ThinnedGridInterpolation", "cubic")
    - **setfileoption**("grb", "InitialTimeCoordinateType" \
      , "Numeric")
  - reading/writing Binary
    - **setfileoption**("bin", "ReadByteOrder", "LittleEndian")
    - **setfileoption**("bin", “WriteByteOrder", “BigEndian")

addfile (1 of 3)

- Used to open a **supported** format only

- **f** = **addfile** (**file_name.ext**, **status**)
  - **file_name** => any valid file name; string
  - **ext** => extension that identifies the type of file; string
    - netCDF: **"nc"** or **"cdf"** [read/write]
    - HDF: **"hdf"** , **"hdfeos"**, **"he5"** [read/write]
    - GRIB: **"grb"** , **"grib"** [read only; GRIB1 or GRIB2]
    - CCMHT: **"ccm"** [read only]
    - extension not required to be attached to file
  - **status** [read/write status] **"r"**, **"c"**, **"w"**
  - **f**
    - reference/pointer to a single file; any valid variable name
    - may have attributes (file attributes or global attributes)

http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclFormatSupport.shtml

addfile (2 of 3)

- **Examples: opening a single file**
  - fin   = **addfile** ("0005-12.**nc**"   , "**r**")
  - fout = **addfile** ("./ncOutput.**nc**" , "**c**")
  - fio   = **addfile** ("/tmp/shea/sample.**hdf**" , "**w**")
  - g   = **addfile** ("/dss/dsxxx/Y12345.**grb**", "**r**")

- **Numerous functions to query contents of supported file**
  - getfilevarnames
  - getfilevardims
  - getfilevaratts
  - getfilevardimsizes
  - getfilevartypes
  - isfilevar
  - isfilevaratt
  - isfilevardim
  - isfilevarcoord

```
diri  = "/fs/cgd/data0/shea/ccm/"
fili   = "testCCM"
ext   = ".ccm"
fin    = addfile(diri+fili+ext , " r ")
```

```
varNames = getfilevarnames (fin)
if (isfilevarcoord(fin, "U", "lat") ) then
…
end if
```

addfile: OPeNDAP (3 of 3) Formerly DoDs

- OPeNDAP enabled: Open Source Project for Network Data Access Protocol
  - access a remote file over the internet
  - file must be located on an OPeNDAP server [max 64 files]
  - only certain operating systems are currently OPeNDAP enabled. NCL can perform OPeNDAP operations on supported systems. Some (CDC) require registration.
  - works with addfile, addfiles, and isfilepresent

```
url_cdc = "http://www.cdc.noaa.gov/cgi-bin/opendap/nph-nc/Datasets/"
fPath    = “ncep.reanalysis/pressure/air.1948.nc"
if ( isfilepresent(url_cdc+fPath) ) then
     f              = addfile ( url_cdc + fPath, "r")
     vNames = getfilevarnames(f)
     if ( any (vNames) .eq. ”T”) then
          t = f->T
     end if
end if
```

Open Dap info at ESRL 1

Example: open, read, output netCDF

```
begin          ; optional
;-------------------------------------------------
;open file and read in data
;-------------------------------------------------
   fin    = addfile ("in.nc”, "r")
   u      = fin->U
;-------------------------------------------------
;create reference to output file
;-------------------------------------------------
    fout   = addfile("out.nc" , "c")
;-------------------------------------------------
;add a global attribute to the file
;--------------------------------------------------
    fout@title = "I/O Example 1"
;--------------------------------------------------
;Output variable u to netCDF file
;---------------------------------------------------
    fout->U= u
end            ; only if begin is present
```

Note: this method of outputing a netCDF file has simple syntax, but can be slow

Reading Binary/ASCII data

- **7 functions for reading binary:**
  - **fbinrecread**: reads multiple unformatted sequential records [Fortran; ieee]
  - **fbinnumrec**: returns the number of unformatted sequential records [Fortran; ieee]
  - **fbindirread**: reads specified record from a Fortran direct access file [ieee]
  - **fbinread**: same as **fbinrecread** but reads only one ieee rec
  - **craybinrecread**: like fbinrecread but for COS blocked data
  - **craybinnumrec**: like fbinnumrec but for COS blocked data
  - **cbinread**: read binary created via C block IO function "write"

- **1 function for reading ASCII data:**
  - **asciiread** **[contributed.ncl: readAsciiTable]**
  - use Fortran/C to read complicated ASCII files

- **all above functions allow data to be shaped**
  - x = **fbinrecread** ("foo_ieee", rnum, (/10,20,30/), "float")
  - a = **asciiread** ("foo_ascii", (/64,128/) , "float")

Include these files

- load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
- load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
- load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
- load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/shea_util.ncl"

- Documentation

Writing Binary/ASCII data

- **4 procedures for writing (ieee) binary data**
  - **fbinrecwrite**: write unformatted fortran sequential recs
  - **fbindirwrite**: write specified record; fortran direct access
  - **fbinwrite**: write a binary file containing a single record
  - **cbinwrite**: write binary file ; mimics C block IO "write“

- **setfileoption:** can be used to alter behavior

- **2 procedures to write ascii data**
  - **asciiwrite**: write a file containing ASCII characters
    - writes a single flat ASCII file. One value per line.
    - No user control of format
  - **write_matrix**: write a multi-dim array to std out or to a file
    - user has format control … pretty-print
    - options for title and row numbering

- use Fortran/C to write complicated ASCII files.

netCDF,GRIB,HDF ==> binary

```
  fin    = addfile ("in.nc", "r")     ; .grb   .hdf   hdfeos
  u      = fin->U
  v      = fin->V
  t      = fin->T
  fout  = "out.bin"
  system ("/bin/rm –f  "+fout)
;-------------------------------------------------------------
; output binary: –1 means append to previous rec
;-------------------------------------------------------------
 setfileoption("bin", “WriteByteOrder", “BigEndian")

  fbinrecwrite (fout, -1, fin->time)
  fbinrecwrite (fout, -1, fin->lev)
  fbinrecwrite (fout, -1, fin->lat)
  fbinrecwrite (fout, -1, fni->lon)
  fbinrecwrite (fout, -1, u)
  fbinrecwrite (fout, -1, v)
  fbinrecwrite (fout, -1, t)
```

binary ==> netCDF

```
; read in data
   lat    = fbinrecread ("./in.bin", 2,  64, "double")
   lon    = fbinrecread ("./in.bin", 3,128, "double")
   u      = fbinrecread ("./in.bin", 6, (/64,128/),"double")

   lat!0                = "lat"
   lat@long_name        = "latitude"
   lat@units            = "degrees_north"
   lon!0                = "lon"
   lon@long_name        = "longitude"
   lon@units            = "degrees_east"

   u!0      = "lat“                          ; named dimensions
   u!1      = "lon"
   u&lat    =  lat                           ; coordinate variables
   u&lon    =  lon
   u@long_name = "zonal wind“   ; attributes
   u@units          = "m/s"

   fout          = addfile ("out.nc", "c")   ; output file
   fout@title = "Binary-to-netCDF"           ; file attribute
   fout->U       = u                         ; write variable to file
```

<u>Example</u>
1

DP Example: multi-formatted data

- **ISCCP**: HDF, binary: 20+yrs, 3hrly: **80**GB [type byte]
  - calculations, regrid, monthly statistics => netCDF, plot
    - per yr: wc= 25.5h, usr=14.9h, sys=8.1h, **24**GB [total **480**GB]

- **NCEP**: GRIB: (same) 20+yrs, 6hrly: **25**+GB
  - calculations, regrid, monthly statistics => netCDF, plot

- **CAM**: netCDF: (same) 20yrs:
  - ensemble of model runs
  - calculations, monthly statistics => netCDF, plot

- **Science**: datasets, calculations, graphics => **paper**

Data Processing Outline

- Algebraic/logical expression operators
- Manual and automatic array creation
- **if** statements , **do** loops
- Built-in and Contributed functions
- User developed NCL functions/procedures
- User developed external procedures
- Sample processing
- Command Line Arguments [CLAs]
- Fortran external subroutines
- NCL as a scripting tool [time permitting]
- Global Variables [time permitting]

Algebraic Operators

Algebraic expression operators

-	Negation	**^**	Exponentiation
*****	Multiply	**/**	Divide
%	Modulus [integers only]	**#**	Matrix Multiply
+	Plus	**-**	Minus
>	Greater than selection	**<**	Less than selection

- Use **(…)** to circumvent precedence rules
- All support scalar and array operations [like f90]
- **+** is overloaded operator
  - algebraic operator:
    - 5.3 + 7.95 ➔ 13.25
  - string concatenator:
    - "alpha" + (5.3 + 7) ➔ "alpha12.3

Logical Expressions

- Similar to f77

Logical expressions formed by relational operators

.le.	(less-than-or-equal)	
.lt.	(less-than)	
.ge.	(greater-than-or-equal)	
.gt.	(greater-than)	
.ne.	(not-equal)	
.eq.	(equal)	
.and.	(and)	
.xor.	(exclusive-or)	**"one or the other but not both."**
.or.	(or)	
.not.	(not)	

Manual Array Creation

- **array constructor characters (/…/)**
  - a_integer = (/1,2,3/)
  - a_float = (/1.0, 2.0, 3.0/) , a_double = (/1., 2, 3.2 /)
  - a_string = (/"abc",”12345",”hello, world"/)
  - a_logical = (/True, False,True/)
  - a_2darray = (/ (/1,2,3/), (/4,5,6/), (/7,8,9/) /)

- **new** function [Fortran dimension, allocate and C malloc]
  - x = **new** (array_size/shape, type, **_FillValue**)
    - **_FillValue** is optional [assigned default if not user specified]
    - **“No_FillValue”** means no missing value assigned
  - a = **new**(3, float)
  - b = **new**(10, double, **1d+20**)
  - c = **new**((/5, 6, 7/), integer)
  - d = **new**(**dimsizes**(U), string)
  - e = **new**(**dimsizes**(**ndtooned**(U)), logical)

- **new** and (/…/) can appear anywhere in script
  - new is not used that often

Automatic Array Creation

- **variable to variable assignment**
  - **y = x** **y** => same size, type as **x** plus meta data
  - no need to pre-allocate space for **y**

- **data importation via supported format**
  - u = f->U
  - same for subset of data: u = f->U(:, 3:9:2, :, 10:20)
    - meta data (coordinate array will reflect subset)

- **functions**
  - return array: **no need** to pre-allocate space
  - T42 = **f2gsh** (gridi, **(/** 64,128**/)**, 42) interpolation func.
  - gridi(10,30,73,144) ➔ T42(10,30,64,128)
    - T42 = **f2gsh_Wrap** (gridi, **(/** 64,128**/)**, 42) ; contributed.ncl

Array Dimension Reduction

- **let T(12,64,128)**
  - Tjan = T(0, :, :) ➔ Tjan(64,128)
  - Tjan automatically becomes 2D: Tjan(64,128)
  - array rank reduced; considered ‘degenerate’ dimension
  - all applicable meta data copied

- **can override dimension reduction**
  - Tjan = T(0:0,:,:) ➔ Tjan(1,64,128)
  - TJAN = **new**((/1,64,128/), **typeof**(T), T@_FillValue)
    - TJAN(0,:,:) = T(0,:,:)

- **Dimension Reduction is a "feature" [really ☺]**

Array Syntax/Operators

- **similar to f90/f95**
- **arrays must be same size and shape: conform**
- **let A and B be (10,30,64,128)**
  - C = A+B
  - D = A-B
  - E = A*B
  - C, D, E automatically created if they did not previously exist
- **let T and P be (10,30,64,128)**
  - theta = T*(1000/P)^0.286 ➔ theta(10,30,64,128)   **Example**
- **let SST be (100,72,144) and SICE = -1.8 (scalar)**
  - SST = SST > SICE   [f90: where (sst.lt.sice) sst = sice]
  - the operation performed by < and > is (sometimes) called *clipping*
- **use built-in functions whenever possible**
  - let T be (10,30,64,128) and P be (30) then
  - theta = T*(1000/conform(T,P,1))^0.286
- **all array operations automatically ignore _FillValue** ****

if statements

Quick

- **if-then-end if** (note: end if has space)

```
if ( all(a.gt.0.) ) then
          …statements
end if
```

- **if-then-else-end if**

```
if ( any(ismissing(a)) ) then
          …statements
else
          …statements
end if
```

no else if

- lazy expression evaluation [left-to-right]

```
if ( any(b.lt.0.)  .and.  all(a.gt.0.) ) then
   …statements
end if
```

loops

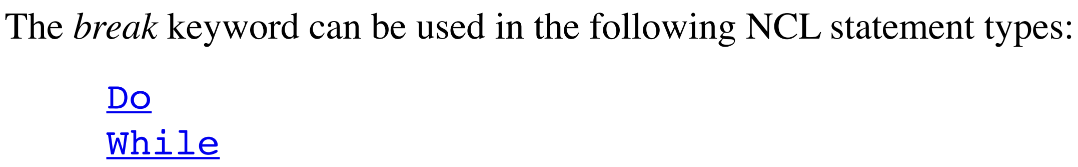
- **do loop** (traditional structure; **end do** has space)
  - do i=*scalar_start_exp, scalar_end_exp [, scalar_skip_exp]*

```
do n = 0, N-1 [,stride]
    … statements
end do                        ; 'end do' has a space
```

  - if start > end
    - identifier 'n' is decremented by a positive stride
    - stride must always be present when start>end

- **do while loop**

```
    do while (x .gt. 100)
              … statements
  end do
```


The *break* keyword can be used in the following NCL statement types:

Do
While

- **break**: loop to abort [f90: exit]
- **continue**: proceed to next iteration [f90: cycle]

- **minimize loop usage in any interpreted language**
  - use array syntax, built-in functions, procedures
  - use Fortran/C codes when efficient looping is required

Built-in Functions and Procedures(1 of 2)

- **use whenever possible**
- **learn and use utility functions**
  - all, any, conform, ind, ind_resolve, dimsizes
  - fspan, ispan, ndtooned, onedtond,
  - mask, ismissing, where
  - system, systemfunc [use local system]
- **functions may require dimension reordering**
  - *must* use named dimensions to reorder

```
;compute zonal and  time average of variable T(time,lev,lat,lon)
;             (zonal average requires rectilinear grid)
; dim_avg works on rightmost dimension
; no meta data transferred
     Tzon = dim_avg( T )                                ; Tzon(time,lev,lat)
     Tavg = dim_avg( T(lev|:, lat|:, lon|:, time|:) )   ; reorder
                                                        ; Tavg(lev,lat,lon)
     Tavg = dim_avg_n( T, 0)                            ; no reorder
```

Built-in Functions and Procedures(2 of 2)

- **functions: NO need to preallocate memory**
  - y = **wgt_runave** (x, wgt, 0)
  - if returning to pre-existing array: must conform

- **procedures: MUST preallocate memory with new**

```
psi = new ( dimsizes(u) , typeof(u) )
chi = new ( dimsizes(u) , typeof(u) )
uv2sfvpg(u,v,psi,chi)
```

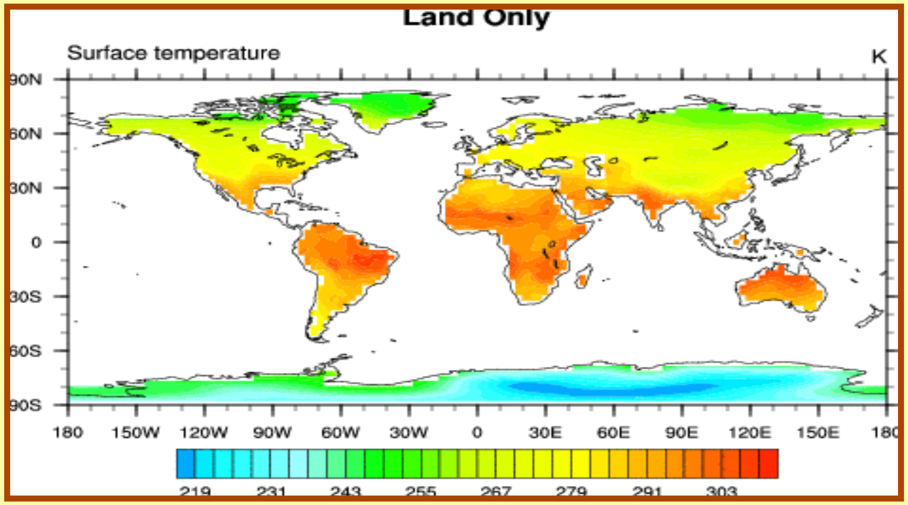
- **functions may be imbedded, procedures can not**
  - keep code simple: avoid 'deep' imbedding

```
; example of a deep imbed
x = f2gsh( fo2fsh( fbinrecread(f,6,(/9,18,72,144/), “float”)),(/nlat,mlon/),42)
; without deep imbedding
G   =  fbinrecread(f, 6, (/1,18,72,144/), "float")
g42 = f2gsh( fo2fsh(G), (/nlat,mlon/),42)
delete (G)
```

mask

- sets values to _FillValue that **DO NOT** equal mask array

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
    in   = addfile(“atmos.nc","r")
    ts   = in->TS(0,:,:)
    oro = in->ORO(0,:,:)
; mask ocean
;  [ocean=0, land=1, sea_ice=2]
    land_only    = ts
    land_only    = mask(ts,oro,1)
end
```



- NCL has 1 degree land-sea mask available [landsea_mask]
    - load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/shea_util.ncl”
    - flags for ocean, land, lake, small island, ice shelf