**Introduction**

Dennis Shea & Rick Brownrigg

NCAR is sponsored by the National Science Foundation

---

**Workshop Overview**

**Objective**
- comfortable with NCL
- minimize learning curve
- access, process and visualize your data
- workshop will *not* make you an expert

**Lots of information**
- language syntax; variable structure
- data formats & manipulation; graphics; processing

**Learning**
- labs very important
- “osmosis” method of learning does not work
- learning any language can be frustrating

**We are here to help you**
Outline: Introduction to NCL

- What is NCL?
- How to run it / what’s required to run it
- The NetCDF data model → NCL’s data model
- Language Syntax
  - Array indexing, coordinate variables, attributes, data types
  - control-of-flow constructs
- Printing
- Debugging

What is NCL?

NCAR Command Language
An Integrated Processing Environment

NetCDF 3/4
HDF-EOS 2/5
HDF 4/5
GRIB 1/2
Shapefile
ASCII  CCM
Binary

Input  Compute  Output

Fortran / C

X11  PS  EPS
PDF  SVG  PNG
NCGM

NetCDF 3/4  HDF
ASCII  Binary
What is NCL?

NCL – the programming language

```ncl
begin
  ;--- data.asc has 6 columns and 500 rows of data.
  data = ascii_read("./data.asc",(/500,6/),"float")
  x = data(:,1)
  y = data(:,4)  ; Read the fifth column of data

  ; smooth the data
  y_smooth = runave(y,25,0)

  data_all = new((/2,dimsizes(y)/),"float")
  data_all(0,:) = y
  data_all(1,:) = y_smooth

  ; create plot
  wks = gsn_open_wks("png","gsn_xy")  ; send graphics to PNG file
  res@tiMainString = "An xy plot Example"  ; title
  res@tiYAxisString = "Dust (ppm)"  ; y axis title
  res@tiXAxisString = "Time"  ; x axis title
  res@xyLineColors = (/"black","red"/)  ; line colors
  res@xyLineThicknesses = (/1,0.2,0/)  ; line thicknesses
  res@xyDashPatterns = (/0,0,0/0/)  ; line patterns

  plot = gsn_xy(wks,x,data_all,res)  ; Draw an XY plot with 1 curve.
end
```

What is NCL?

NCL – the interpreter

- Ncl is an interpreted language (not compiled)
  - Like python, R, matlab
  - Unlike C, C++, fortran

- Many operations implemented in C/fortran under the hood = fast performance

- Interpreter is invoked via the “ncl” command:
  ```ncl
  $ ncl
  ```
Why NCL?

Pros
- **Support** netCDF-3/4, HDF-4/5, and GRIB-1/2
- **Common data structure**: NetCDF model
- **Many** useful & unique *Application Functions*
- **Publication quality** graphics out of the box
- **Consistent documentation**
- **Many examples** to get you started
- **Excellent Support**

Cons
- Interactive environment is rather **crude**
- **No debugger** at user level

Useful? .... **YES!!! VERY much so!!**

What is required to use NCL?

- Unix-like operating system
- **Linux**, **fedora**, **CentOS**, **Mac OS X**
- **Basic command-line skills**
- Text editor
Running NCL: Interactive

```bash
$ ncl
Copyright (C) 1995–2015 - All Rights Reserved
University Corporation for Atmospheric Research
NCAR Command Language Version 6.4.0
The use of this software is governed by a License Agreement.
See http://www.ncl.ucar.edu/ for more details.
ncl 0> pi = 4.0 * atan(1.0)
ncl 1> print(pi)

Variable: pi
Type: float
Total Size: 4 bytes
1 values
Number of Dimensions: 1
Dimensions and sizes: [1]
Coordinates:
(0) 3.141593
ncl 2> quit
$
```

Useful for *simple* testing – otherwise not recommended

Running NCL: Batch mode

```bash
# run an NCL "script" stored in a file...
$ ncl myScript.ncl
# regular shell operators apply...
$ ncl <myScript.ncl       # I/O redirection
$ ncl myScript.ncl >myScript.out  # capture output to a file
$ ncl myScript.ncl &     # run as background process
```

- Recommended over interactive mode
- Typical programming work flow (true of any language!):
  - Edit
  - Run/test/debug
  - Repeat
Running NCL: command-line structure

$ ncl -h
Usage: ncl -fhnopxQV <args> <file.ncl>
- f: use new file structure and NetCDF4 features when possible
- h: print this message and exit
- n: don't enumerate values in print()
- o: retain former behavior for certain backwards-incompatible changes
- p: don't page output from the system() command
- x: echo NCL commands
- Q: turn off echo of NCL version and copyright info
- V: print NCL version and exit

- Some options modify behavior
- Others print information and exit
- May be combined: ncl –np myScript.ncl

Understanding the NetCDF data model

- Why is this important – NCL’s variable model is based upon NetCDF’s variable model.
  - NCL makes GRIB, HDF, HDF-EOS look like NetCDF
  - This consistent and uniform view of disparate file formats is a very powerful feature!

- NetCDF is a structured, binary, file format.
  - Can’t view contents directly from command-line
  - Use tools like ncdump or ncl_filedump

- Is inherently intended for array-like data
  - Commonplace in atmospheric science and other science and engineering disciplines that employ finite-element methods.

- Comprised of variables, dimensions, attributes
Parts of a NetCDF file: global attributes

- Attributes are arbitrary name/value pairs
- *Global* attributes are metadata about the *file*

```
$ nclFiledump 80.nc
path:  80.nc
file global attributes:
  Conventions : NCAR-CSM
  source : Data converted from CCM History Tape Format
  case : b020.05
  title : b020.05 CSM1.2 (1980-2000 ramp,tauvis=.01, s04 dir,SCYC, 6hr data)
  hybrid_s sigma_pressure :
  Pressure at a grid point (lon(i),lat(j),lev(k)) is computed
  using the formula:
  p(i,j,k) = A(k)*P0 + B(k)*PS(i,j)
where A, B, P0, and PS are contained in the variables whose
names are given by the attributes of the vertical coordinate
variable A_var, B_var, P0_var, and PS_var respectively.
```

Parts of a NetCDF file: dimensions

- N-dimensional arrays have sizes, or “shape”
- Dimensions are *names* for array sizes in the file
- Are integer values
- One dimension may be “unlimited”:
  - arrays allowed to grow along that dimension
  - all other dimensions are fixed

```
$ nclFiledump 80.nc
dimensions:
  time = 1  // unlimited
  lat = 64
  lon = 128
  lev = 18
```
Parts of a NetCDF file: variables

- Variables are arrays of data
- Have *type*, *shape* (dimensionality), *attributes* (metadata)

```
dimensions:
    time = 1 // unlimited
    lev = 23
    lat = 128
    lon = 256
variables:
    integer lev(lev)
        positive : down
        units : hPa
        long_name : pressure

    double time(time)
        info : first day of the month
        calendar : gregorian
        long_name : initial time
        units : hours since 1800-01-01 00:00

    float T(time, lev, lat, lon)
        cell_method : time: mean
        standard_name : air_temperature
        units : K
        long_name : Temperature
```

NetCDF/NCL: Coordinate Variable (CV)

- **CV**: Coordinate Variable definition
  - one *dimensional* variable
  - *dimension name* is the same as the variable name
  - must be numeric (integer, float, double)
  - must be monotonic (increasing or decreasing)
- **CV examples**: `variable_name(dimension_name)`
  - `lat(lat)`, `lon(lon)`, `plevel(plevel)`, `time(time)`
- Usage: `variable temp(lat, lon)`
  - `temp(i,j)` is temperature at i'th, j'th element
  - latitude of that element is value at `lat(i)`
  - longitude of the element is value at `lon(j)`
- `Q(time, plevel, lat, lon)`
  - **CV**: `Q(:, {925:400}, {-20:60}, {130:280})`
  - Index: `Q(:, 3:10, 24:40, 42:75)`
NetCDF Conventions

**Convention:** set of rules for file contents
- makes data comparison easier
- facilitates development of viewing (eg: ncview) & processing tools (NetCDF Operators; Climate Data Op.)

**COARDS** (1995; frozen)
- Cooperative Ocean/Atmosphere Research Data Service
- created for rectilinear grids
- [http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html](http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html)

**CF** (2005/2006; continues to evolve)
- Climate and Forecast Metadata Convention (1.0 -> 1.6)
- generalizes and extends the COARDS convention
- much more complex; curvilinear and unstructured grids
- calendar attributes (eg: no_leap, 360_day, 365_day, ...)

Most climate related data archives use NetCDF and adhere to these conventions: eg: CMIP5, CMIP3, CESM, IPCC. etc

---

**NCL variables look like NetCDF variables**

- **array** is basic element [ length 1 (scalar)]
- (may have) additional information: **not required**

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.35</td>
</tr>
<tr>
<td>4.36</td>
</tr>
<tr>
<td>9.73</td>
</tr>
<tr>
<td>17.01</td>
</tr>
<tr>
<td>-0.63</td>
</tr>
<tr>
<td>-4.29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x(2,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.46</td>
</tr>
<tr>
<td>10.39</td>
</tr>
<tr>
<td>1.76</td>
</tr>
<tr>
<td>-1.43</td>
</tr>
<tr>
<td>5.85</td>
</tr>
</tbody>
</table>

- **name**: x
- **type**: float [real]
- **shape**: 2-dimensions
- **size**: 6 (rows) x 5 (columns) [ row major; C, Matlab]
- **values**: x(2,3) = 8.46 [ 0-based indexing; C ]
- **long_name**: "Temperature"
- **units**: "degC"
- **_FillValue**: -99.99

- **named dimensions**: x(time, lat)
- **lat**: (/ -60, -30, 0, 30, 60 /)
**Detailed Look at an NCL Variable**

```ncl
0 > f = addfile("UV300.nc", "r")  ; read nc, grb, hdf, hdfeos
1 > u = f->U  ; import variable (STRUCTURE)
2 > printVarSummary (u)  ; variable overview
3 > printMinMax (u, 0)  ; variable Min & Max values
```

**Variable: u**
- **Type:** float
- **Total Size:** 65536 bytes
- **16384 values**
- **Number of Dimensions:** 3
- **Dimensions and Sizes:** `[time|2] x [lat | 64] x [lon | 128]`
- **Coordinates:**
  - `time`: `[1 .. 7]`
  - `lat`: `[-87.8638 .. 87.8638]`
  - `lon`: `[0 .. 357.185]`
- **Number of Attributes:** 5
  - `_FillValue`: `1e36`
  - `units`: `m/s`
  - `long_name`: Zonal Wind
  - `short_name`: U
  - `missing_value`: `1e36`

(0) **Zonal Wind (m/s)**: `min=-15.2682  max=55.7283`

---

**NetCDF [NCL] Variable model**

```
p = f->SLP
```

NCL reads
- **data values**
- **attributes**
- **coordinate arrays**

as a **single** data object.

---

**NCL syntax/funcs**
- **query**
- **use**
- **modify**
- **add**

**any aspect of variable**

---

**Variable: p**
- **Type:** float
- **Total Size:** 29272320 bytes
- **7318080 values**
- **Number of Dimensions:** 3
- **Dimensions and sizes:** `[time | 252] x [latitude | 121] x [longitude | 240]`
- **Coordinates:**
  - `time`: `[780168..963504]`
  - `latitude`: `[90..-90]`
  - `longitude`: `[0 .. 358.5]`
- **Number of Attributes:** 4
  - `FillValue`: `1e+20`
  - `units`: hPa
  - `long_name`: Mean sea level pressure
  - `missing_value`: `1e+20`

---

**Classic netCDF Variable Model**

```
printVarSummary(p)
```

**Output**

---

(0) **Mean sea level pressure (hPa)**: `min=998.648  max=1023.25`

---

10
NCL: control constructs

if (a.eq.b) then
  ...
else
  ...
end if

do i=0,99
  ...
end do

do while(a.gt.0)
  ...
end do
Built-in and user-defined functions

Built-in functions

User defined

function getTwoPi()
begin
  return 2*4*atan(1)
end

NCL Syntax Characters (subset)

 Assignment
:= - reassignment
;
- comment [can appear anywhere; text to right ; ignored]

/- ;/ - comment block; typically text spanning multiple lines

-> - use to (im/ex)port variables via addfile(s) function(s)

@ - access/create attributes
!
- access/create named dimension
&
- access/create coordinate variable
{...} - coordinate subscripting

$...$ - enclose strings when (im/ex)port variables via addfile(s)

(//) - array construction (variable); remove meta data

[..] - list construction;
[::] - all elements of a list

- array syntax
|
- separator for named dimensions
\ - continue character [statement to span multiple lines]
:: - syntax for external shared objects (eg, fortran/C)
. - access elements of a hdf5 (compound) group
Two Fundamental Data Structures

• Arrays
  - Have fixed shape and size
  - All elements are of the same type
  - Scalars are 1D arrays of size 1
  - Constant time to access an element

• List:
  - Can grow/shrink dynamically
  - Can contain elements of differing types
  - Access time proportional to size of list

Simple Variable Creation

```plaintext
a_int        = 1
a_float     = 2.0 ; 0.00002 , 2e-5
a_double = 3.2d ; 0.0032d , 3.2d-3
a_string   = "a"
a_logical  = True [False] ; note capital T/F
```

• array constructor characters (/…/)
  - a_integer = (/1, 2, 3/) ; ispan(1,3,1)
  - a_float = (/2.0, 5, 8.0/) ; fspan(2,8,3)
  - a_double = (/12, 2d0, 3.2 /) ; (/12,2 ,3.2 /)*1d0
  - a_string = ("abcd", "e", "Hello, World")
  - a_logical = (/True, False, True/)
  - a_2darray = (/ (/1,2,3/), (/4,5,6/), (/7,8,9/) /)
Variable Creation and Deletion

```
a  = 2.0
pi = 4.0*atan(1.0)
s = ('Melbourne', 'Sydney', 'Toulouse', 'Boulder')
r = f->precip ; (time,lat,lon)
R = random_normal(20,7, (N,M) ) ; R(N,M)
q = new ( (ntim, klev, nlat, mlon), "double")
; free memory; Generally, do not need to do this
; delete each variable individually
delete(a)
delete(pi)
delete(s)
delete(r)
delete(R)
; delete multiple variables in one line
delete( [a, pi, s, r, R, q] ) ; [/…/] list syntax
```

Data Types

**numeric (classic netCDF3)**
- double (64 bit)
- float (32 bit)
- long (64 bit; signed +/-)
- integer (32 bit; signed +/-)
- short (16 bit; signed +/-)
- byte ( 8 bit, signed +/-)
- complex NOT supported

**numeric (netCDF4; HDF5)**
- int64 (64 bit; signed +/-)
- uint64 (64 bit; unsigned )
- uint (32 bit; unsigned )
- ulong (32 bit; unsigned )
- ushort (16 bit; unsigned )
- ubyte ( 8 bit, unsigned)

**non-numeric**
- string
- character
- graphic
- file
- logical
- list

**list**
[/ vari, vard, vars, …/]
Conversion between data types

- **NCL** is a ‘strongly typed’ language
  - constraints on mixing data types
- **coercion**
  - implicit conversion of one type to another
- **automatic coercion when no info is lost**
  - let i be integer and x be float or double
    - fortran: \( x = i \) and \( i = x \)
    - NCL: \( x = i \) and \( i = \text{toint}(x) \)
- **many functions to perform conversions**

Variable Reassignment

- **NCL := will not allow the following**
  \[ k = (/ 1, 3, 4, 9 /) \] ; 1d array, type integer
  \[ \text{... later in code ...} \]
  \[ k = (/ 17.5, 21.4 /) \] ; Error! different size and type

- **Two approaches**
  - 2 steps
    - delete(k) ; delete existing variable
    - \( k = (/ 17.5, 21.4 /) \) ; new assignment
  - version 6.1.2
    - \( k := (/ 17.5, 21.4 /) \) ; delete previous variable
      ; and reassign ‘\( k \)’

- **NCL := allows dynamic variable subsetting**
  \[ x := x(:,4,:, :) \] ; same variable
Array semantics

\[ a = (12, 32, 18, 17) \]
\[ b = (9, 23, 15, 8) \]
\[ s = 0.25 \quad \text{; a scalar quantity} \]

; add two arrays, mult. by a scalar
\[ c = s \times (a + b) \]

; c is (5.25, 13.75, 8.25, 6.25)

\[ \begin{align*}
\text{do } i &= 0, 99 \\
& \quad \text{do } j = 0, 255 \\
& \quad \quad c(i, j) = s \times (a(i, j) + b(i, j)) \\
& \quad \text{end do} \\
& \text{end do}
\end{align*} \]

Variable Subscripting

**Standard Array Subscripting (Indexing)**

- ranges: start/end and [optional] stride
- index values separated by :
- omitting start/end index implies default begin/end

Consider \( T(\text{time, lat, lon}) \)

\[ \begin{align*}
T & \rightarrow \text{entire array [ don't use } T(:,:,:) \text{ ]} \\
T(0,:,:)5 & \rightarrow 1^{\text{st}} \text{ time index, all lat, every 5^{th} lon} \\
T(:,::-1,:50) & \rightarrow 1^{\text{st}} 4 \text{ time indices, reverse, 1^{st} 51 lon} \\
T(7:12,45,10:20) & \rightarrow 6 \text{ time indices, 46^{th} value of lat, 10-20 indices of lon}
\end{align*} \]

Good programming: Use variables not hard wired #

\[ T(\text{tstrt:tlast, : , ln1:ln2 }) \rightarrow \text{time index } tstrt:tlast, \text{ all lat ; , longitude index values ln1:ln2} \]
**Arrays: Indexing & Dimension Numbers**

- **row major**
  - left dimension varies **slowest**; right dim varies **fastest**
  - dimension numbering **left to right** [0,1,..]
- **subscripts**
  - 0-based [ entire range for N index values: 0,N-1 ]

Consider $T(:, :, :, :) \rightarrow T(0, 1, 2, 3)$

- **left** dimension is 0 : varies slowest
- **mid-left** dimension is 1
- **mid-right** dimension is 2
- **right** dimension is 3 : varies fastest

- Some processing functions operate on dimension numbers
  - Example: $T(ntim, klev, nlat, mlon) \rightarrow T(0, 1, 2, 3)$
  - $Tzon = \text{dim\_avg\_n}(T, 3) \rightarrow Tzon(ntim, klev, nlat)$
  - $Tstd = \text{dim\_stddev\_n}(T, 0) \rightarrow Tstd(klev, nlat, mlon)$

**NCL – Fortran/Matlab/R Array Indexing**

Different language/tool ordering. There is no ‘right/wrong’

- **NCL/C/C++/py** : 0-based; left (slowest) - right (fastest)
- **fortran, Matlab, R** : 1-based; left (fastest) - right(slowest)
- **IDL** : 0-based; left (fastest) - right(slowest)

Ex: Our logical view of a 2D array is a matrix:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

But computer memory is linear:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>w</td>
</tr>
<tr>
<td>y</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

NCL: 0,0 0,1 0,2 0,3 1,0 1,1 1,2 1,3
Fortran 1,1 2,1 3,1 4,1 1,2 2,2 2,3 2,4
**Variable Subscripting**

**Coordinate Variable Subscripting**
- **only** applies to coordinate variables (1D, mono)
- same rules apply for ranges, strides, defaults
- use curly brackets {...}
- standard and coordinate subs can be mixed
  [if no reorder]

\[
T(2:7,\{-30:30\},:) \Rightarrow \text{six times, all lon, lat -30° to +30° (inclusive)}
\]

\[
T(0,\{-20\},\{-180:35:3\}) \Rightarrow \text{1st time, lat nearest -20°, every 3rd lon between -180 and 35°}
\]

\[
T(::12,\{\text{latS:latN}\},:) \Rightarrow \text{all times/lon, lat latS to latN (inclusive)}
\]

\[
T(8,\{\text{latS}\},\{\text{lonL:lonR}\}:3) \Rightarrow \text{9th time, lat nearest latS, every 3rd lon between latL and lonR}
\]

---

**Subscripting: Index, CV**

Grid of two dimensional data

**by index:**

\[T(9:13,1:8)\]

**by coordinate:**

\[T(\{-10:20\},\{-170:-110\})\]

**combined:**

\[T(\{-10:20\}, 1:8)\]
Classification of NCL Variables

• All NCL variables follow the NetCDF model
  - e.g., have type, dimensions, attributes, etc.

• Special subclasses of NCL variables:
  - *File* variables: f->U notation
  - *Graphics* variables
  - *Coordinate* variables: & notation
  - *List variables*: [/ ... /] and [:] notation

• Some functions expect instances of a particular variable type (filevars, listvars, etc.)

List Variables

• List variables can hold heterogeneous types
• Appear as a 1-D array
• Can grow/shrink dynamically
• Alternatively viewed as a stack (lifo) or a queue (fifo)

```ncl
i = (/ (/1,2,3/), (/4,5,6/) /) ; 2D integer array
x = 5.0 ; scalar of type float
d = (/100000.d, 283457.23d/) ; 1D double array
s = "abcde" ; string
c = stringtochar("abcde") ; character
vl = [/1, x, d, c, s/] ; construct list via [/.../]

my_list = NewList("lifo") ; or NewList("fifo")
ListPush(my_list,x)
ListPush(my_list,y)
ListPush(my_list,s)

cnt = ListCount(my_list)
print(cnt) ; prints "3"
```
List Variables

- Access list members by index or “head” of list
- Tests for membership and list-length

```fortran
; access a list element by direct index
do i=0, ListCount(my_list)-1
   print(my_list[i])
end do

; test for membership in a list
idx = ListIndex(my_list, x)

; access "head" of the list
; - last item to be "pushed" for lifo
; - first item to be "pushed" for fifo
do while(ListCount(my_list).gt.0)
   print(ListPop(my_list)) ; removes element
end do
```

Metadata access

- **Syntax to access ncl-variable components**
  - attributes: @ (numeric, text)
  - named dimensions: ! (text)
  - coordinates: & (numeric)

- @, !, & can be used to assign/modify/retrieve

- most frequently, metadata is read from files
Attributes [ @ ]

- **info associated with a variable or file**
  - attributes can be any data type except file or list
  - scalar, multi dimensional array (string, numeric)

- **assign/access with @ character**
  
  ```
  T       = (/ 10, 25, 39 /)
  T@units = “degC”
  T@long_name = “Temperature”
  T@wgts  = (/ 0.25, 0.5, 0.25 /)
  T@_FillValue = -999
  title    = T@long_name
  ```

- **attribute functions [isatt, getfilevaratts]**
  
  ```
  if (isatt(T,"units")) then .... end if
  atts  = getfilevaratts (fin, "T")
  ```

  - **delete** an attribute: `delete(T@wgts)`

.BatchNorm

_FILLValue attribute

- **Unidata & NCL reserved attribute; CF compliant**

- **netCDF Operators [NCO] & CDO: _FillValue attribute**
  - ncview: recognizes `missing_value` attribute (COARDS)
    - best to create netCDF files with both

- **NCL functions recognize _FillValue**
  - most functions will ignore for computations (eg, "avg")
  - use built-in function "ismissing" to check for _FillValue
    - if (any (ismissing(T)) ) then … end if
    - **NOTE:** if (any(T.eq.T@_FillValue)) will not work

- **NCL: best to not use zero as a _FillValue**
  - OK except when contouring [random bug]
NCL (netCDF): Named Dimensions

• may be “named”: \( x(\text{time}, \text{level}, \text{lat}, \text{lon}) \)
• dimensions are named on netCDF files
  - alternative way to reference subscripts

• assigned with ! character \{let \( T(:,:,:) \) -> \( T(0,1,2) \)}
  - \( T!0 = \"time\" \) ; leftmost [slowest varying] dim
  - \( T!1 = \"lat\" \)
  - \( T!2 = \"lon\" \) ; rightmost [fastest varying] dim

• dim names may be renamed, retrieved
  - \( T!1 = \"latitude\" \) … \( \text{dName} = T!2 \)
• delete can eliminate: delete \((T!2)\)

• named dimensions used to reshape
  - \( T(\text{lat}(:,), \text{lon}(:,), \text{time}(:)) \)

Create, Assign Coordinate Variables

• create 1D array
  - \( \text{time} = (/1980, 1981, 1982/) \)
  - \( \text{time}@\text{units} = \"yyyy\" \)
  - \( \text{lon} = \text{ispans}(0, 355, 5) \)
  - \( \text{lon}@\text{units} = \"degrees_east\" \)

• assign dimension name [same as variable name]
  - \( \text{time}!0 = \"time\" \)
  - \( \text{lon}!0 = \"lon\" \)

• let \( x(:, :) \) … dimension numbers \( x(0,1) \)
• name dimensions
  - \( x!0 = \"time\" \) … \( x!1 = \"lon\" \)
• assign coordinate variables to \( x \)
  - \( x&\text{time} = \text{time} \) … \( x&\text{lon} = \text{lon} \)
Variable Subscripting

Named Dimensions
- only used for dimension reordering
- indicated by |
- dim names must be used for each subscript
- named/coordinate subscripting can be mixed

Consider \( T(\text{time}, \text{lat}, \text{lon}) \)
\[
t = T(\text{lat}|, \text{lon}|, \text{time}|) \quad \Rightarrow \quad t(\text{lat}, \text{lon}, \text{time})
\]
\[
t = T(\text{time}|, \{\text{lon}|90:120\}, \{\text{lat}|-20:20\}) \quad \Rightarrow \quad \text{all times, 90-120° lon, -20-20 lat}
\]

“printing”

- **printVarSummary** \((u)\)
  - gross overview of a variable

- **printMinMax** \((u, 0)\)
  - Zonal Wind (m/s) : min= -15.27 max= 55.73

- **print(...)**
  - includes same info as **printVarSummary**
  - prints each value, with associated index info
  - \((32, 56, 118) \quad 3.339051 \quad \ldots \) one value/line

- **write_matrix(...)**
  - formatted tabular output of numerical data

- **print_table(...)**
  - formatted tabular output of mixed types
**printVarSummary**

- **Print overview of variable contents**
  - type (e.g., float, string), dimension information
  - coordinate information (if present)
  - attributes (if present)

- **printVarSummary** (u)

<table>
<thead>
<tr>
<th>Variable: u</th>
<th>Type: double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Size: 1179648 bytes</td>
<td>147456 values</td>
</tr>
<tr>
<td>Number of Dimensions: 4</td>
<td>Dimensions / Sizes: [time</td>
</tr>
<tr>
<td>Coordinates:</td>
<td></td>
</tr>
<tr>
<td>time:</td>
<td>[4046..4046]</td>
</tr>
<tr>
<td>lev:</td>
<td>[4.809 .. 992.5282]</td>
</tr>
<tr>
<td>lat:</td>
<td>[-87.86379 .. 87.86379]</td>
</tr>
<tr>
<td>lon:</td>
<td>[ 0. 0 .. 357.1875]</td>
</tr>
<tr>
<td>Number of Attributes: 2</td>
<td></td>
</tr>
<tr>
<td>long_name: zonal wind component</td>
<td></td>
</tr>
<tr>
<td>units: m/s</td>
<td></td>
</tr>
</tbody>
</table>

---

**print** (1 of 3)

- **Prints out all variable information including**
  - All meta data, values
  - T(lat,lon): print (T)

<table>
<thead>
<tr>
<th>Variable: T</th>
<th>Type: float</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Size: 32768 bytes</td>
<td>8192 values</td>
</tr>
<tr>
<td>Number of Dimensions: 2</td>
<td>Dimensions / Sizes: [lat</td>
</tr>
<tr>
<td>Coordinates:</td>
<td></td>
</tr>
<tr>
<td>lat:</td>
<td>[-87.86379 .. 87.86379]</td>
</tr>
<tr>
<td>lon:</td>
<td>[ 0. 0 .. 357.1875]</td>
</tr>
<tr>
<td>Number of Attributes: 2</td>
<td></td>
</tr>
<tr>
<td>long_name: Temperature</td>
<td></td>
</tr>
<tr>
<td>units: degC</td>
<td></td>
</tr>
<tr>
<td>(0,0)</td>
<td>-31.7</td>
</tr>
<tr>
<td>(0,1)</td>
<td>-31.4</td>
</tr>
<tr>
<td>(0,2)</td>
<td>-32.3</td>
</tr>
<tr>
<td>(0,3)</td>
<td>-33.4</td>
</tr>
<tr>
<td>(0,4)</td>
<td>-31.3 etc. [entire T array will be printed]</td>
</tr>
</tbody>
</table>
Variable: T (subsection)
Type: float
Total Size: 256 bytes
   64 values
Number of Dimensions: 1
   Coordinates:
      lat: [-87.86379 .. 87.86379]
Number of Attributes: 3
   long_name: Temperature
   units: degC
   lon: 109.6875 [ added ]

(0) -40.7
(1) -33.0
(2) -25.1
(3) -20.0
(4) -15.3 etc.

print (2 of 3)
• can be used to print a subset of array
  - meta data, values
  - T(lat,lon): print(T(:,103)) or print(T(:,{110}))

print (3 of 3)
• print with embedded strings
  - no meta data
  - print("T: min=
      min(T) + " max="
      max(T))

(0) T: min=-53.8125 max=25.9736

• sprintf and sprinti provide formatting
  - often used in graphic labels
  - print( "min(T) = " + sprintf("%5.2f ", min(T)) )

(0) min(T) = -53.81

• sprinti can left fill with zeros (ex: let n=3)
  - fname = "h" + sprinti("%0.5i", n) + ".nc"
  - print("file name = " + fname)

(0) file name = h00003.nc
write_matrix(x[\*][\*], fmt, opt)

- pretty-print 2D array (table) to standard out
  - integer, float, double
  - user format control (fmt)
  - T(N,M), N=7, M=5: \texttt{write_matrix(T, "5f7.2", False)}

<table>
<thead>
<tr>
<th></th>
<th>4.35</th>
<th>4.39</th>
<th>0.27</th>
<th>-3.35</th>
<th>-6.90</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.36</td>
<td>4.66</td>
<td>3.77</td>
<td>-1.66</td>
<td>4.06</td>
</tr>
<tr>
<td></td>
<td>9.73</td>
<td>-5.84</td>
<td>0.89</td>
<td>8.46</td>
<td>10.39</td>
</tr>
<tr>
<td></td>
<td>4.91</td>
<td>4.59</td>
<td>-3.09</td>
<td>7.55</td>
<td>4.56</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>3.68</td>
<td>5.08</td>
<td>0.14</td>
<td>-5.63</td>
</tr>
<tr>
<td></td>
<td>-0.63</td>
<td>-4.12</td>
<td>-2.51</td>
<td>1.76</td>
<td>-1.43</td>
</tr>
<tr>
<td></td>
<td>-4.29</td>
<td>0.07</td>
<td>5.85</td>
<td>0.87</td>
<td>8.65</td>
</tr>
</tbody>
</table>

- can also create an ASCII file

```python
opt = True
opt@fout = "foo.ascii" ; file name
write_matrix(T, "5f7.2", opt)
```

print_table(listvar, fmt)

- pretty-print a list-variable to standard output
- write formatted reports or CSV files

```python
a = (/ 111, 222, 333, 444 /)
b = (/ 1.1, 2.2, 3.3 /)
c = (/ "a", "b", "c" /)
d = (/ 11h, 22h /)
f = (/111, 221, 331, 441, 551, 661/)
alist = [/ a, b, c, d /]
print_table(alist, "%d,%16.2f,%s,%d,%ld")
  111,
  1.10,a,11,11
  222,
  2.20,b,22,22
  333,
  3.30,c,,33
  444,
  ,,,44
  ,
  ,,,55
  ,
  ,,,66
```
Debugging, Error Messages, Help

• NCL does not have a built-in debugger
  • use `print /printVarSummary` ; examine output!
    • `nmsg = num( ismissing( x ) )` ; count # _FillValue
    • `print(“x: min=“+min(x) +” max=“+max(x) )`

• Error messages; **Warning** or **Fatal**
  • Look at the message; often describe problem/issue
    • *eg*: Fatal: left and right side have different sizes
    • `printVarSummary` of variables before Fatal

• Common error messages:
  
  http://www.ncl.ucar.edu/Document/Language/error_messages.shtml

NCL Support

• **Documentation and Examples**
  
  - [http://www.ncl.ucar.edu/](http://www.ncl.ucar.edu/)
    • numerous downloadable examples to get you going
  - downloadable reference manuals [pdf], FAQ

  **ncl-talk@ucar.edu** (users must subscribe)

  [http://www.ncl.ucar.edu/Support/ncl_talk.shtml](http://www.ncl.ucar.edu/Support/ncl_talk.shtml)
  
  • Two Modes: (a) email-by-email, (b) digest
  • include enough info to facilitate answering
  • do *not* attach large files (> 1.5 Mb) they will be rejected, use ftp/web
  • do *not* ‘dump’ a messy script to ncl-talk
  • Our time is valuable too!