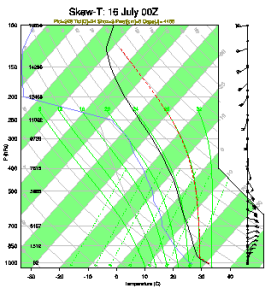
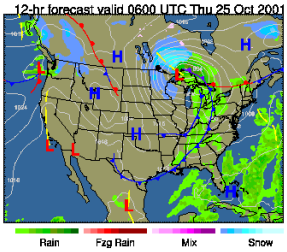
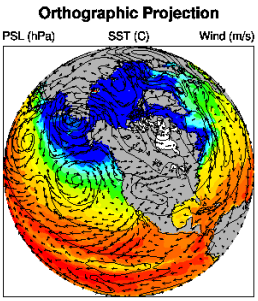
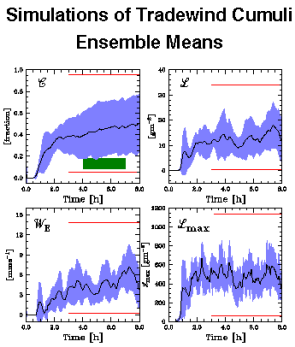
Introduction

Dennis Shea









NCAR is sponsored by the National Science Foundation

Workshop Overview

Objective

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

Labs important: you must invest the time

- "osmosis" method of learning does not work

NCL Support

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

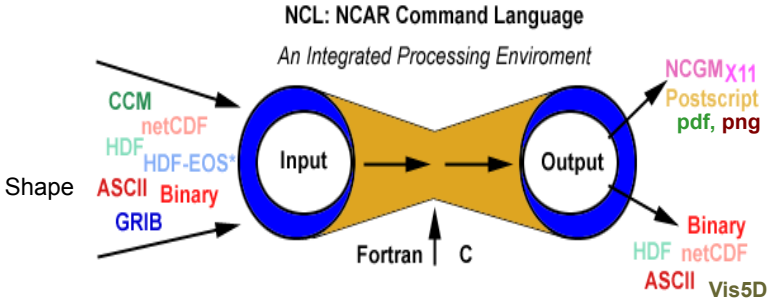
ncl-talk@ucar.edu

users must subscribe

http://www.ncl.ucar.edu/Support/ncl_talk.shtml

NCL Overview

- **Integrated data processing environment**



- **freeware: supported, public domain**
- **portable: linux/unix, windows (cygwin), MacOS**
- **general purpose: unique capabilities**
- **excellent 2D graphics** (limited 3D)
- **parallel: IO, computation** (alpha version fall 2012)

Executing NCL: Interactive (1 of 3)

- **Interactive Mode (Command line)**
  - **ncl** *[options][command-line-arguments]* <return>
    ncl> enter commands
    ncl> **quit** <return>
  - can save (record) interactive commands
    ncl> **record** "file_name"
    ncl> …. enter commands …
    ncl> **stop record**

- **Interactive environment**
  - use for simple testing
  - can use 'up/down arrow' to recall previous lines
  - **not** as 'friendly' as (say) IDL, Matlab, ferret
    - not good at error recovery

Running NCL: Batch (2 of 3)
Recommended

- **Batch Mode** [**.ncl** suffix is optional]
  - **ncl** *[options][arguments]* **script**.ncl
    - **ncl <** script.ncl [also acceptable]
  - **ncl** *[options][arguments]* **script.**ncl **>&!** out
  - **ncl** *[options][arguments]* **script.**ncl **>&!** out **&**
    - appending "**&**" means put in background
    - note: the **>&!** **&** are **csh** and **tcsh** syntax

- **NCL built for larger processing tasks**
  - better accomplished via a script **(recommended)**
    - use editor (vi, nedit, emacs, ..) to open/modify
    - enter/delete statements
    - run the script

netCDF / NCL Relationship

- NCL **variable** model is **based** on the netCDF variable model
- NCL makes GRIB, HDF, HDF-EOS look like netCDF files

netCDF Conventions

Convention: set of accepted rules for file contents
- make data comparison easier
- facilitate automatic use of viewing (eg: **ncview**)

COARDS (1995; frozen)
- **C**ooperative **O**cean/**A**tmosphere **R**esearch **D**ata **S**ervice
- http://ferret.wrc.noaa.gov/noaa_coop/coop_cdf_profile.html

GDT (1999; frozen)
- **G**regory-**D**rach-**T**aylor
- Precursor to **CF**

CF (2005/2006; continues to evolve)
- **C**limate and **F**orecast Metadata Convention (1.0 -> 1.6)
- generalize and extend the **COARDS** convention

Parts of netCDF file

ncdump –h foo.nc (or **ncl_filedump** foo.nc)

DIMENSION SIZES & NAMES
dimensions:
lat = 64
lon = 128
time = 12

time=UNLIMITED (12 currently)

FILE ATTRIBUTES
global attributes:
title = "Temp: 1999"
source = "NCAR"
Conventions = "CF-1.0"

exercise:
ncl_filedump FOO.nc | less
ncl_filedump FOO.grb | less

VARIABLES:
Names , Types, Attributes, Coordinate Variables
variables:
float lat(lat)
lat:long_name = "latitude"
lat:units = "degrees_north"
float lon(lon)
lon:long_name = "longitude"
lon:units = "degrees_east"
double time(time)
time:long_name = "time"
time:units = "hours_since …"
float T(time, lat, lon)
T:long_name = "Temperature"
T:units = "degC"
T:**missing_value** = 1.e+20f
T:**_FillValue** = 1.e+20f

NCL/netCDF Variable Semantics

double T(time, lat, lon)
T: long_name = "Temperature"
T: units = "degC"
T: _FillValue = 1.e+20f

variable type – double (float, int, short,..)
variable name – T
named dimensions – time, lat, lon
attributes – long_name, units, _FillValue

Donnees de desagregation produites par le Cerfacs

float prr(**time, y, x**)

prr:_FillValue = -9999.f ;	***CF***
prr:missing_value = -9999.f ;	***COARDS***
prr:long_name = "Liquid Precipitation" ;	***CF, COARDS***
prr:grid_mapping = "Lambert_Conformal" ;	
prr:units = "kg m-2 s-1" ;	***CF, COARDS***
prr:height = "surface" ;	
prr:coordinates = "**lon lat**" ;	***CF***

netCDF/NCL variable

- **array** [could be of length 1 (scalar)]
- **(may) have additional information**

x

```
  4.35    4.39    0.27   -3.35   -6.90
  4.36    4.66    3.77   -1.66    4.06
  9.73   -5.84    0.89    8.46   10.39
    17    3.68    5.08    0.14   -5.63
 -0.63   -4.12   -2.51    1.76   -1.43
 -4.29    0.07    5.85    0.87    8.65
```

name: x
type: float [real]
shape: 2-dimensions
size: 6 (rows) x 5 (columns)
values: x(2,3) = 8.46 [row major, 0-based indexing]

long_name: "Temperature"
units: "degC"
named dimensions: x(time,lat)
lat: (/ -60, -30 ,0, 30, 60 /)
time: (/2000, 2001, 2002, 2003, 2004, 2005, 2006 /)

Meta data

Detailed Look netCDF Variable (NCL)

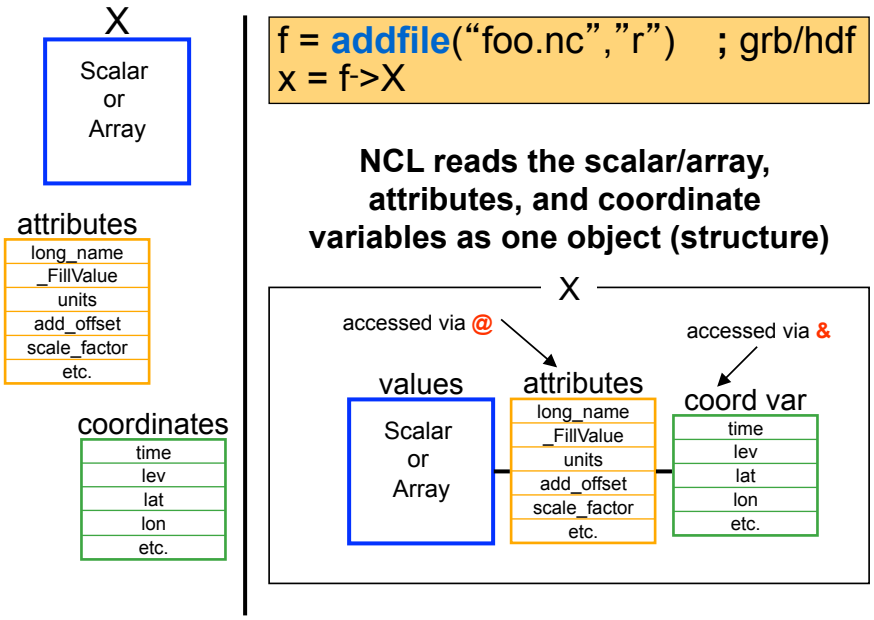
```
ncl <return>                                  ; interactive mode
ncl 0 > f = addfile ("UV300.nc", "r")         ; open file (nc, grb, hdf, hdfeos)
ncl 1 > u = f->U                              ; import STRUCTURE
ncl 2 > printVarSummary (u)                   ; overview of variable
```

```
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          [CF ]
   units :           m/s           [COARDS, CF]
   long_name :       Zonal Wind    [COARDS, CF]
   short_name :      U
   missing_value :   1e36          [COARDS; CF-1.6 ]
```

Classic netCDF Variable Model

NCL syntax/funcs
query
use
modify
add
any aspect of data object

netCDF [NCL] Variable model




```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
```

B+W graphic

```
f = addfile("erai_1989-2009.mon.msl_psl.nc","r")   ; open file [hdf,grib]
p = f->SLP                                         ; (time,lat,lon)
                                                   ; ( 252,121,240)

printVarSummary(p)                                 ; netCDF variable model

wks  = gsn_open_wks("ps","parvis_1")               ; open a PS file
plot = gsn_csm_contour_map(wks,p(0,:,:),False)     ; default plot
                                                   ; uses attributes, coordinates
```

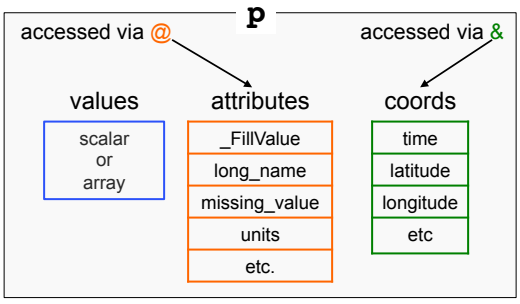
NetCDF [NCL] Variable model

`p = f->SLP`

NCL reads

- *data values (scalar or array)*
- *attributes* @
- *coordinate variables* &

as a **single** data object.



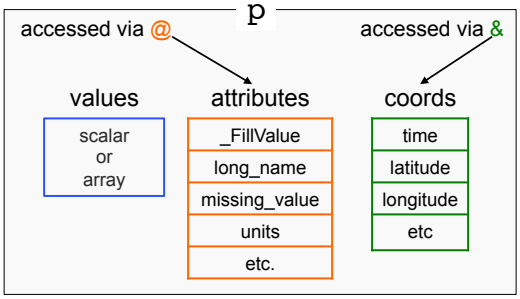
NetCDF [NCL] Variable model

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

NCL reads

- *data values*
- *attributes*
- *coordinate arrays*

as a **single** data object.



"**printVarSummary**(p)" output

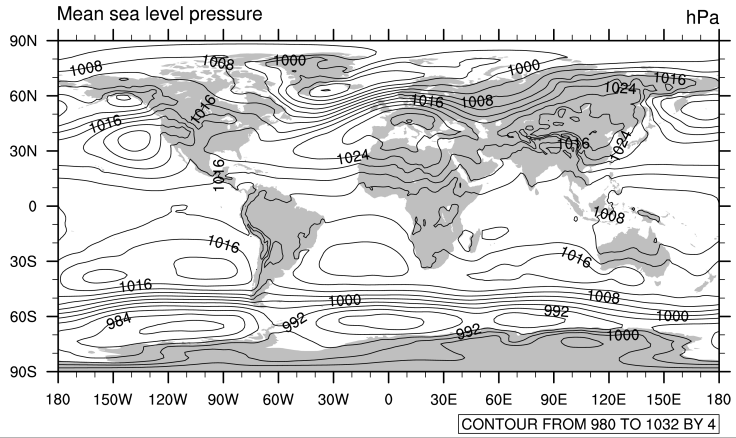
```
Variable: p
Type: float
Total Size: 29272320 bytes
            7318080 values
Number of Dimensions: 3
Dimensions and sizes:     [time | 252] x [latitude | 121] x [longitude | 240]
Coordinates:
            time: [780168..963504]
            latitude: [90..-90]
            longitude: [ 0..358.5]
Number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

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

f = addfile("erai_1989-2009.mon.msl_psl.nc","r")   ; open file [hdf,grib]
p = f->SLP                                         ; (time,lat,lon)
                                                   ; ( 252,121,240)

printVarSummary(p)                                 ; netCDF variable model

wks  = gsn_open_wks("ps","parvis_1")               ; open a PS file
plot = gsn_csm_contour_map(wks,p(0,:,:),False)     ; default B&W plot
```



Language Basics

NCL Syntax Characters (subset)

- **=** - assignment
- **;** - comment [can appear anywhere]
- **->** - 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)

Data Types

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

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

enumeric (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)

snumeric
[numeric , enumeric]

Simple Variable Creation

```
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 = (/1. , 2 , 3.2 /) *1d5
  - a_string = (/"abcd", "e", "Hello, World"/)
  - a_logical = (/True, False, True/)
  - a_2darray = (/ (/1,2,3/), (/4,5,6/), (/7,8,9/) /)

http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclDataTypes.shtml#BasicNumericTypes

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
```

Conversion between data types

- **NCL** is a 'strongly typed' language
- **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=toint(x)
- **many functions to perform conversions**

Meta Data

- **information associated with variable or file**
  - **attributes: @** (numeric, text)
  - **named dimensions: !** (text)
  - **coordinates: &** (numeric)
- **@**, **!**, **&** can be used to **create/assign, retrieve**
- most frequently, **meta data 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@x2d            = (/  (/1,2,3/),  (/4,5,6/), (/7,8,9/) /)
T@_FillValue   = -999
title              = x@long_name
```

- **attribute functions** [isatt, getfilevaratts]

```
if (isatt(T,"units")) then ....  end  if
atts  = getfilevaratts (fin, "T")
```

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

_FillValue attribute

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

- **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

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

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

Arrays

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

Consider T(:,:,:) ➔T (**0**,**1**,**2**)
left dimension is **0**
middle dimension is **1**
right dimension is **2**

Different language/tool ordering:
- **NCL/C/C++** : 0-based; left/right -> slowest/fastest
- **fortran**, **Matlab**, **R**: 1-based; left/right -> fastest/slowest
- **IDL** : 0-based; left/right -> fastest/slowest

NCL (netCDF): Named Dimensions

- **may be "named"**
- provides alternative way to reference subscripts
  - recommendation: always name dimensions

- **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 = "LAT" … dName = T!2
- **delete** can eliminate: delete (T!2)

- **named dimensions used to reshape**
  - T(lat|:, lon|:, time|:)

Create and Assign Coordinate Variables

- **create 1D array**
  - time = (/ 1980, 1981, 1982 /)
  - time@units = "yyyy"
  - lon = **ispan**(0, 355, 5)
  - lon@units = "degrees_E"

- **assign dimension name** [same as variable name]
  - time!0 = "time"
  - lon!0 = "lon"

- let **x(:,:)** **, name dimensions**
  - x!0 = "time" … x!1 = "lon"
- **assign coordinate variables to x**
  - x&time = time … x&lon = lon

netCDF/NCL: Coordinate Variable (CV)

- **CV**: **C**oordinat**e** **V**ariable (Unidata definition)
  - **one dimensional** variable in which **dimension name** is the same as the **variable name**
  - must be monotonically increasing or decreasing
- Examples of **CV**
  - lat(lat), longitude(longitude), plevel(plevel), time(time)
- **CV** allow 'natural' coordinates via **{…}** syntax
  - Q(time,plevel,lat,longitude)
    - CV: Q(**:**, **{**925:400**}**, **{**-20:60**}**, **{**130:280**}**)
    - Index: Q(**:**, 3:10, 24:40, 42:75)

The following is **not** a coordinate variable:

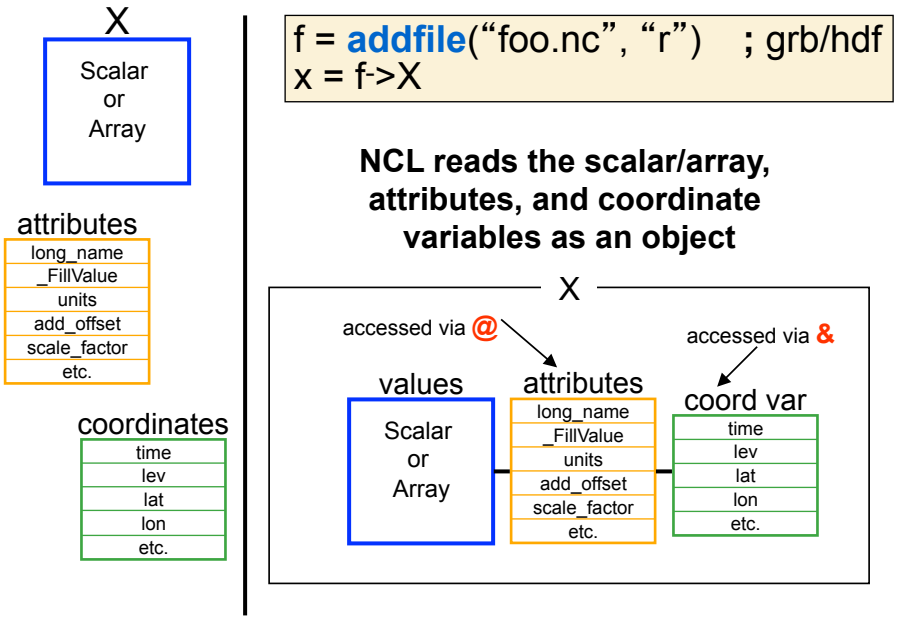
float xlat(nlat, mlon) ; **two**-dimensions
xlat:units = "degrees_north"

It is an array that contains coordinate information.
Requires use of standard integer index values

Meta Data Syntax Review: Access/Change/Create/Delete

- **@ attribute**
  - u@long_name = "U"
  - lonName = u@long_name
- **! named dimensions**
  - u!0 = "TIME"
  - tName = u!0
- **& coordinate variable**
  - u&lat = (/ -90., -85, ... , 85., 90. /)
  - latitude = u&lat
- **$ substitute string**
  - x = fin->$variable(n)$ … x = fin->$"T: p"$

netCDF [NCL] Variable model



Variable Subscripting (1 of 3)

Standard Array Subscripting (Indexing)

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

Consider T(time,lat,lon)

T	➔	entire array [don't use T(:,:,:)]
T(0,:,::5)	➔	1st time index, all lat, every 5th lon
T(:3, ::-1, :50)	➔	1st 4 time indices, reverse, 1st 51 lon
T(7:12,45,10:20)	➔	6 time indices, 46th value of lat, 10-20 indices of lon

Good programming: use variables to index:

T(tstrt:tlast, : , ln1:ln2) ➔ time index tstrt:tlast, all lat :, longitude index values ln1:ln2

Variable Subscripting (2 of 3)

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},:) ➔ six times, all lon, lat -30° to +30° (inclusive)

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

T(::12,{**latS:latN**},:) ➔ all times/lon, lat latS to latN (inclusive)

T(8,{**latS**},{**lonL:lonR**:3}) ➔ 9th time, lat nearest **latS**, every 3rd lon between **latL** and **lonR**

Variable Subscripting (3 of 3)

Named Dimensions

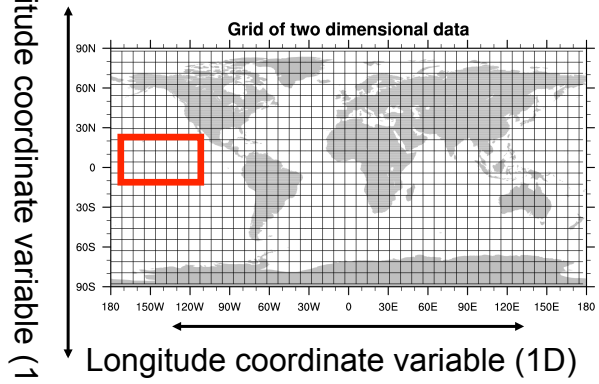
- **only** used for dimension reordering
- indicated by **|**
- dim names must be used for each subscript
- named/coordinate subscripting can be mixed

Consider T(time,lat,lon)

t = T(lat|:, lon|:, time|:) ➔ makes **t**(lat,lon,time)

t = T(time|:,{lon|90:120},{lat|-20:20}) ➔ all times, 90-120° lon, -20-20° lat

Subscripting: Index, CV



Standard:

T(9:13,1:8)

Coordinate:

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

Combined:

T({-10:20}, 1:8)

"printing"

- **printVarSummary**
  - gives gross overview of a variable
- **print**
  - includes same info as **printVarSummary**
  - prints each value
- **write_matrix**
  - print to standard out or a file
  - format control of numerical output
  - can write to file also

printVarSummary

- **Print overview of variable contents**
  - type
  - dimension information
  - coordinate information (if present)
  - attributes (if present)
- **printVarSummary** (u)

```
Variable: u
Type: double
Total Size: 1179648 bytes
            147456   values
Number of Dimensions:  4
            Dimensions / Sizes:  [time | 1]  x  [lev | 18]  x  [lat | 64] x  [lon | 128]
Coordinates:
            time:   [4046..4046]
            lev:    [4.809 .. 992.5282]
            lat:    [-87.86379 .. 87.86379]
            lon:    [ 0. 0 .. 357.1875]
Number of Attributes:  2
            long_name: zonal wind component
            units:          m/s
```

print (1 of 3)

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

```
Variable: T
Type: float
Total Size: 32768 bytes
            8192   values
Number of Dimensions:  2
            Dimensions / Sizes: [lat | 64] x  [lon | 128]
Coordinates:
            lat:    [-87.86379 .. 87.86379]
            lon:    [ 0. 0 .. 357.1875]
Number of Attributes:  2
            long_name: Temperature
            units:          degC
(0,0)  -31.7
(0,1)  -31.4
(0,2)  -32.3
(0,3)  -33.4
(0,4)  -31.3    etc. [entire T array will be printed]
```

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}))

```
Variable: T (subsection)
Type: float
Total Size: 256 bytes
            64   values
Number of Dimensions:  1
            Dimensions / Sizes: [lat | 64]
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 (3 of 3)

- **print with embedded strings**
  - no meta data
  - **print** ("min(T)="+**min**(T)+" max(T)="+**max**(T))

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

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

```
(0)  min(T) = -53.81
```

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

```
(0)  file name = h00003.nc
```