NCL Basics I

Language

Karin Meier-Fleischer, DKRZ

Contents

- Interactive mode
- Special syntax characters
- Expressions: Algebraic and logical operators
- Data types
- Variables
- Attributes
- Named dimensions and coordinate variables
- Statements and loops
- print and printVarSummary
- Create and run a NCL script

Interactive mode

Start NCL

```
>  ncl
Copyright (C) 1995-2014 - All Rights Reserved
University Corporation for Atmospheric Research
NCAR Command Language Version 6.2.1
The use of this software is governed by a License
Agreement.
See http://www.ncl.ucar.edu/ for more details.
ncl 0>
```

Exit NCL

```
ncl 1>  quit
```

Special syntax characters

=	assignment syntax
:=	reassignment operator
;	begins a comment
->	use to im- or export variables via addfile function
@	create or reference attributes
!	create or reference named dimension
&	create or reference coordinate variable
$...$	enclose strings when im- or export variables via addfile
(/…/)	array construction
{…}	coordinate subscripting
[…]	list construction
:	array syntax, e.g. b(:) means all elements of array b
|	seperator for named dimensions
\	continue character
::	syntax for external shared objects (e.g. fortran/C)

Algebraic operators

+	Addition
-	Substarction / Negation
*	Multiplication
/	Division
%	Modulus
>	Greater than
<	Less than
^	Exponentiation
#	Matrix multiplication

Logical operators

.lt.	Less than
.le.	Less equal
.eq.	Equal
.ne.	Not equal
.ge.	Greater equal
.gt.	Greater than
.and.	And
.or.	Or
.xor.	Exclusive or
.not.	Not

Data types

Numeric data types

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

Enumeric data types

int64	(64 bit; signed +/-)
uint64	(64 bit; unsigned)
uint	(32 bit; unsigned)
ulong	(32 bit or 64 bit; unsigned)
ushort	(16 bit; unsigned)
ubyte	(8 bit; unsigned)

Non-numeric data types

string
character
graphic
file
logical
List

Snumeric

numeric, enumeric

Variable creation

```
x      = 0.12                              ; float
y      = -4                                ; integer
z      = 20.d                              ; double
title  = "This is the title string"  ; string
a      = True                              ; logical
```

Array constructor characters (/../)

```
a      = (/1, 2, 3, 4/)                    ; integer array
b      = (/1, 2.0, 3.0, 4./)               ; float array
c      = (/1., 2, 3., 4.0/) * 1d5          ; double array
d      = (/"hello", "world"/)              ; string array
e      = (/True, False, False, True/)  ; logical array
f      = (/ (/1,2/), (/3,4/), (/5,6/) /)  ; 2D array
```

Delete a variable

```
delete(title)                  delete( [/ a,b,c,d,e,f /] )
```

Variable subscripting (1)

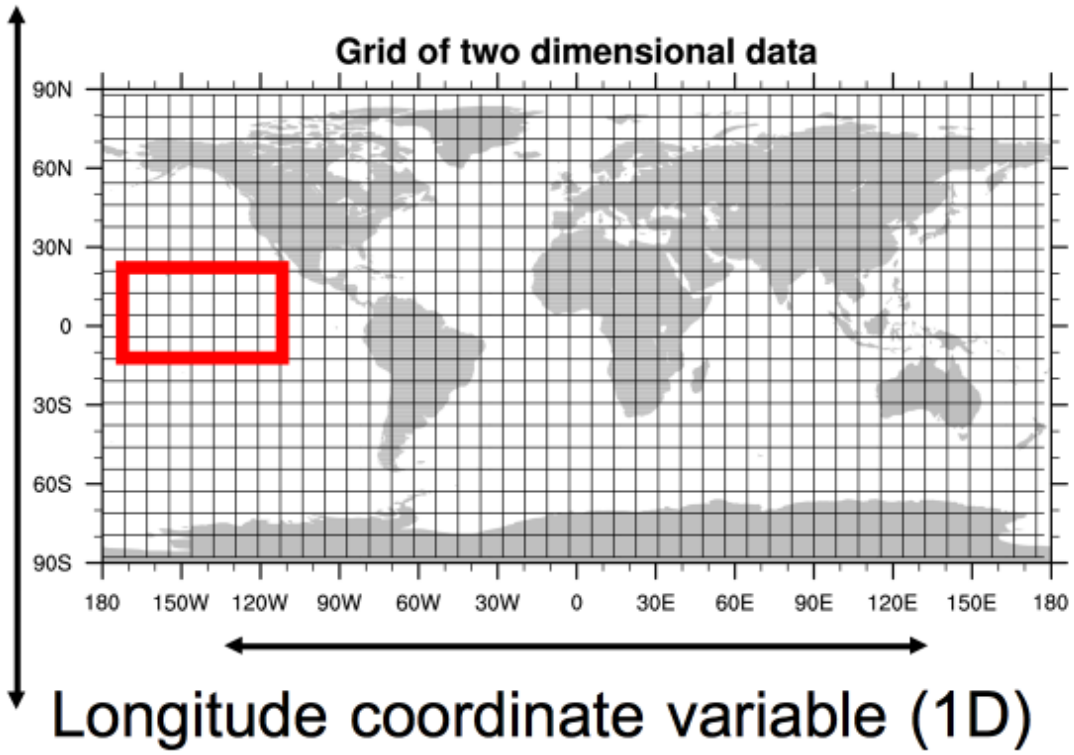
Standard subscript **m:n:I** range **m** to **n** in strides of **i**

```
3D array   slp(time,lat,lon)

slp              →  the entire array
slp(0,:,:)       →  1st timestep, all lat, all lon
slp(:,:,::5)     →  all timesteps, all lat, every 5th lon
slp(:3,15:30,::-1) →  first 4 timesteps, indice 15 to 30
                       of lat, reverse lon
```

Coordinate variable subscripting

```
slp(0,{-45},{-60:60:5})   →  1st timestep, near lat -45°, lon
                              near -60° to 60° in steps of 5
```

Variable subscripting (2)



Standard:

T(9:13,1:8)

Coordinate:

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

Combined:

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

Graphic: NCL workshop from NCAR

Conversion between data types

- Strongly typed language **Temp** is NOT equal to **temp**

- Coercion
  - Implicit conversion of one type to another

- Automatic coercion when no information is lost

```
i = 2
x = 4.5
x = i        →   no information is lost
i = x        →   error
fatal:["NclVar.c":1390]:Assignment type mismatch, right hand side
can't be coerced to type of left hand side
fatal:["Execute.c":8565]:Execute: Error occurred at or near line 2

i = toint(x) →   i = 4
```

- Conversion functions
  - toint, tofloat, todouble, tolong, toshort, tostring, tobyte,……

Attributes

- Info about a variable or file (called meta data)
- Attributes can be of any data type except file or list
- Scalar, multi dimensional array (string, numeric)

To assign an attribute

```
lon@units          = "degrees_east"
t@long_name        = "Near-Surface Air Temperature"
time@units         = "days since 1949-12-01 00:00:00"
temp@_FillValue    = 1e20
temp@missing_value = 1e20
```

Many attribute functions available, e.g.

```
isattr
getfilevaratts
```

Delete an attribute

```
delete(time@units)
```

Attribute _FillValue

- **Reserved attribute (Unidata and NCL)**
- **netCDF-CF convention compliant**
- **Most NCL functions recognize the _FillValue**

If ***missing_value*** is set, the attribute ***_FillValue*** must be the same value. If missing_value is set and _FillValue is undefined, NCL will create it internally

```
slp@_FillValue = slp@missing_value
```

Function ***ismissing*** checks for _FillValue

```
if(any(ismissing(slp)))then
   …do anything…
end if
```

!! Recommended: Do not use zero as a _FillValue

Arrays

- Row major
  - left dimension varies slowest
  - right varies fastest
  - dimension numbering from left to right
- Subscripts are zero based

  N values: subscripts from index **0** to index **N-1**

  e.g. array x has 12 elements the

  first element x(**0**)

  last element x(**11**)

```
T(12,5,4)      left   dimension index 0 with 12 elements
                      (varying slowest)
               middle dimension index 1 with 5 elements
               right  dimension index 2 with 4 elements
                      (varying fastest)
```

NCL/C/C++ **0-based and row major**

Fortran **1-based and column major**

Named dimensions

Shape: number of dimensions
Dimension size: number of elements for each dimension

A single-dimension variable with one value is called a scalar variable.

Dimensions are numbered from left to right 0 to N-1 (like arrays).
consider slp(:,:,:) → slp(0,1,2) → 3 dimensional variable

Create named dimensions for a 3 dimensional variable slp(:,:,:), the following statements will attach the names to the dimensions using the **!** character:

```
slp!0  = "time"
slp!1  = "latitude"
slp!2  = "longitude"
```

Reshape an array tas(lon,lat,time) → tas(time|:, lat|:, lon|:)

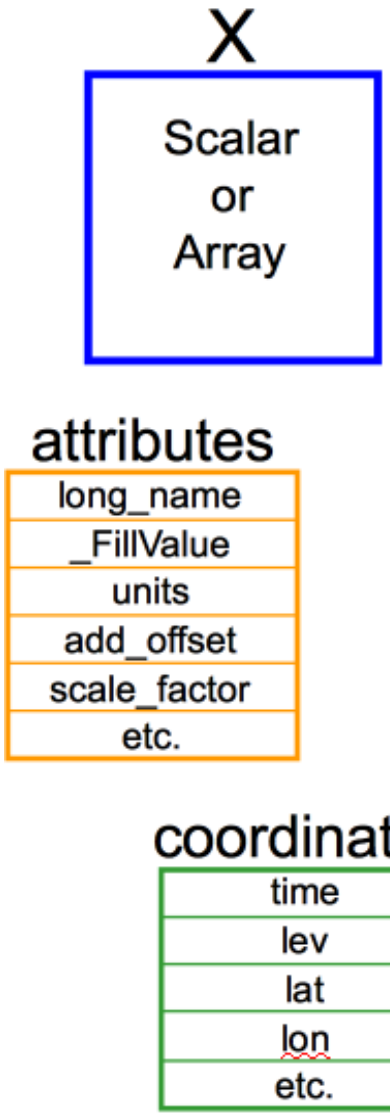
Coordinate variables

- 1D variable with the same name as a dimension, which names the coordinate values of the dimension
- must not have any missing data
- must be strictly monotonic (values increasing or decreasing)

E.g. 2D array **temp(4,5)**

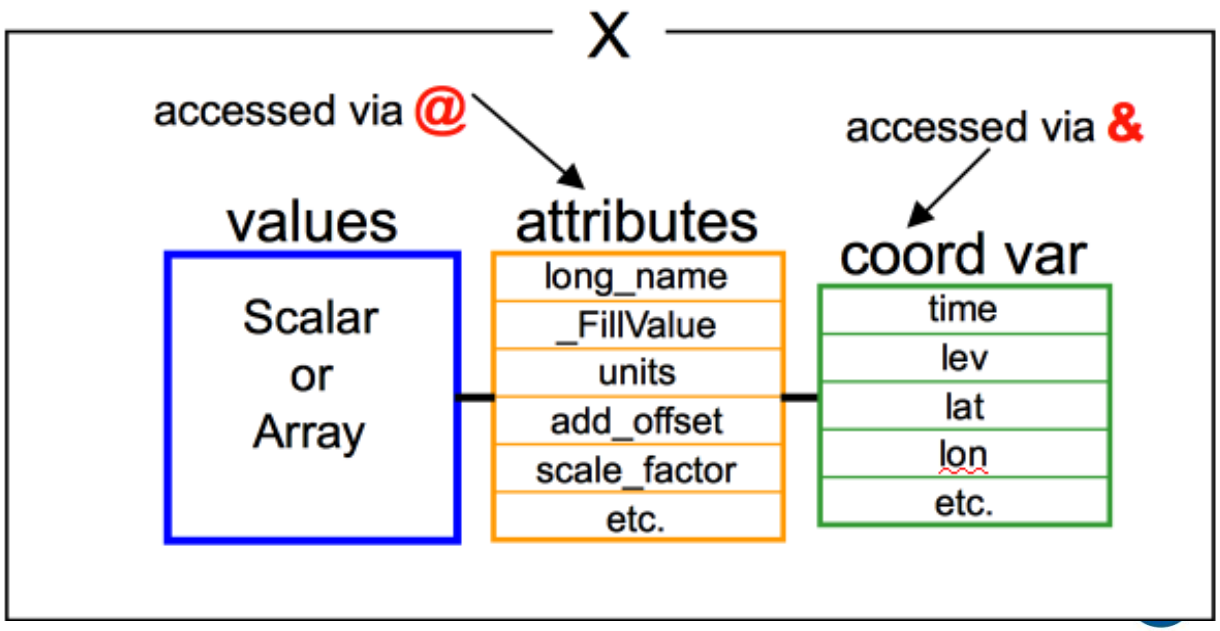
```
temp!0 = "lat"                  ; left dimension
temp!1 = "lon"                  ; right dimension

lon_pts = (/ 0., 15., 30., 45., 60. /)   ; size 5
lat_pts = (/ 30., 40., 50., 60. /)       ; size 4
lon_pts@units = "degrees_east"
lat_pts@units = "degrees_north"

temp&lon = lon_pts
temp&lat = lat_pts
```

NCL: netCDF variable model




```
f = addfile("foo.nc", "r")    ; grb/hdf
x = f->X
```

NCL reads the scalar/array, attributes, and coordinate variables as an object



Graphic: NCL workshop from NCAR

Statements and loops (1)

Statements

- Blocks (a group of statements; begin-end)

```
begin
        statement 1
        statement 2
        ……
end
```

- Conditional expressions (if-then, if-then-else)

```
if (scalar_logical_expression) then
        [statement(s)]
else
        [statement(s)]
end if
```

Statements and loops (2)

Statements

- Loops (do, do-while)

```
do n=start,end[,stride]
      [statement(s)]
end do
```

  - Loop while a logical expression is True:

```
do while (scalar_logical_expression)
      [statement(s)]
end do
```

- Assignments / Reassignments

```
var = "This is a string"              ;-- var of type string
var := (/1.0,10.0,15.0/)              ;-- var of type float
```

Statements and loops (3)

Statements

- Procedures / Functions

```
undef ("procedure_name")
procedure procedure_name(declaration_list)
local local_identifier_list
begin
   statement_list
end

undef ("function_name")
function function_name(declaration_list)
local local_identifier_list
begin
   statement_list
   return(return_value)
end
```

printVarSummary, print, write_matrix (1)

- printVarSummary - information about variable

printVarSummary(tsurf)

```
Variable: tsurf
Type: float
Total Size: 2949120 bytes
            737280 values
Number of Dimensions: 3
Dimensions and sizes:  [time | 40] x [lat | 96] x [lon | 192]
Coordinates:
            time: [   0.. 234]
            lat: [88.57216851400727..-88.57216851400727]
            lon: [-180..178.125]
Number Of Attributes: 5
  long_name :  surface temperature
  units :      K
  code :       169
  table :      128
  grid_type :  gaussian
```

printVarSummary, print, write_matrix (2)

- print - same info as printVarSummary
  - print values

ndims = dimsizes(tsurf)

print(ndims)

Variable: ndims
Type: integer
Total Size: 12 bytes
3 values
Number of Dimensions: 1
Dimensions and sizes: [3]
Coordinates:
(0) 40
(1) 96
(2) 192

print("ndims = "+ndims)

(0) ndims = 40
(1) ndims = 96
(2) ndims = 192

printVarSummary, print, write_matrix (3)

Embeded strings

```
print("Min temp = " + min(temp) + "  Max temp = " + max(temp))
```

→ Min temp = 21.7 Max temp = 37.1

Formated printing

```
print("Value  x = " + sprintf("%5.2f",x))
```

→ Value x = 6.87

Leading zeros

```
fn = "file_" + sprinti("%0.5i",2) + ".nc"
print("file name = "+fn)
```

→ file name = file_00002.nc

printVarSummary, print, write_matrix (4)

Other print functions

`write_matrix(variable,format,opt)`	pretty-print 2D array to stdout
`print_table(list)`	formatted print of all elements from a list
`printMinMax(variable,0)`	prints the minimum and maximum value of a variable
`printFileVarSummary(file,variable)`	prints a summary of a file variable's information

Create and run a NCL script

Create a NCL script, e.g. *myfirstscript.ncl*

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

begin
   print("Hello World")
   x = 1.5
   print(x)
end
```

Run the script

```
ncl myfirstscript.ncl
```

See what happens