NCL quick reference card

NCL version 6.2.1 Karin Meier-Fleischer, DKRZ
November 4, 2014 Mary Haley, NCAR

Syntax characters

=	assignment syntax
:=	reassignment operator
;	starts a comment
@	create or reference an attribute
!	create or reference a named dimension
&	create or reference a coordinate variable
$...$	enclose strings when importing or exporting variables via *addfile*
{...}	subscript arrays using coordinate values
[...]	subscripts variables of type *list*
(/.../)	array constructor
[/.../]	list constructor
:	array syntax delimiter
|	separator for named dimensions
\	continuation character for wrapping long lines
::	separator when calling external codes
->	used to im/export variables from/to supported file formats

Expressions

Algebraic operators

+	Addition, string concatenation
-	Subtraction / Negation
*****	Multiplication
/	Division
%	Modulus (integers only)
>	Greater than
<	Less than
^	Exponentiation
#	Matrix multiplication

Logical operators

.lt.	Less than
.le.	Less than or equal
.eq.	Equal
.ne.	Not equal
.ge.	Greater than or equal
.gt.	Greater than
.and.	AND
.or.	OR
.xor.	Exclusive OR
.not.	NOT

Data types

Numeric

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

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

string
character
graphic
file
logical
list

Variables

Assign a variable

```
x = 1                            ; integer
y = 2.6                          ; float
d = 20.d                         ; double
str = "This is a string"         ; string
res = True                       ; logical (True/False)
a = (/1,2,3,4/)                  ; integer array
b = (/2,7.0,4./)                 ; float array
c = (/1.,2,3.,4.0/) * 1d5        ; double array
d = (/"red","green","blue"/)     ; string array
e = (/True,False,False,True/)    ; logical array
f = (/(/1,2/),(/3,6/),(/4,2/)/) ; 2D array (3 x 2)
```

Arrays

The leftmost dimension (dim) of a multi-dim array varies slowest and the rightmost dim varies fastest (row major).

```
a = (/4,2,1,3/)         ; 4 elements; index 0-3
b = (/0,1,1,0/)         ; 4 elements; index 0-3

c = a + b          →   c = (/4,3,2,3/)
c = a – b          →   c = (/4,1,0,3/)
c = a * b          →   c = (/0,2,1,0/)
c = a/(b+0.1)      →   c = (/40,1.8182, 0.909090,30/)
```

To create a new array

```
n = new(4,integer)       → integer array of size 4
q = new((/2,3,5/),float) → float array of size 2x3x5
l = new(100,float,1e20)  → float array with
                           _FillValue=1e20
cities = new(20,string)  → string array of size 20
```

Standard subscripting of arrays

The indices used in standard subscripting are integers and the general form of a standard subscript is:

```
  m:n:i          ; range m to n in strides of i
```

```
  a = (/1,2,3,4,5,6/)
 a1 = a(3)         ; a1 is 4
 a2 = a(0:2)       ; a2 contains 1,2,3
 a3 = a(0:4:2)     ; a3 contains 1,3,5
 a4 = a(1:4:-1)    ; a4 contains 5,4,3,2
 a5 = a(:3)        ; a5 contains 1,2,3,4
 a6 = a(5:3)       ; a6 contains 6,5,4
 a7 = a(::-1)      ; reverse a 6,5,4,3,2,1
```

Named dimensions

The dimensions of an array are numbered from 0 to *n*-1. To attach a name to an array dimension, use the **!** character.

```
varNew!0 = "time"
varNew!1 = "lev"
varNew!2 = "lat"
varNew!3 = "lon"
```

Named subscripting

Named dimensions allow you to reorder and subscript arrays.

```
pres(lat,lon)       ; lat=21, lon=40

pres_new1 = pres(lon|:, lat|:)  ; reorder (reshape)
pres_new2 = pres(lon|19:38, lat|0:9)
              ; define an new array pres_new2(20,10)
              ; with pres_new2(lon,lat)
```

Coordinate variables

A coordinate variable is a one-dimensional variable with the same name as a dimension, which provides coordinate values for that dimension. It must be strictly monotonic (values increasing or decreasing, not mixed).

```
lat_pts       = (/30.,40.,50.,60.,/)   ; size 4
lon_pts       = (/ 0.,15, 30, 45, 60/)  ; size 5
lat_pts@units = "degrees_north"  ; set units attribute
lon_pts@units = "degrees_east"   ; set units attribute
grid          = new((/4,5/),float)   ; define 2D array
grid!0        = "lat"          ; name left dimension
grid!1        = "lon"          ; name right dimension
grid&lat      = lat_pts        ; assign values to named
                               ; dimension "lat"
grid&lon      = lon_pts        ; assign values to named
                               ; dimension "lon"
```

Coordinate subscripting

For coordinate subscripting, all of the rules for standard subscriptin apply except for curly brackets { }, which are used to distinguish coordinate subscripts from standard subscripts.

```
m     = (/-5.0,10.0,15.0,20.0,25.0,30.0/)
m!0   = "lat"         ; name dimension 0
m&lat = m             ; associate the array
mw    = m({-5. : 25. : 2})  ; contains -5.0,-15.0,25.0
```

Use coordinate subscripting to select a subregion in a global grid.

```
var(96,192)         ; 96 lat and 192 lon elements

var_region = var({20:60},{0:70})
```

→ Returns an array containing latitudes nearest to the values between 20 and 60 degrees inclusive, and longitudes nearest to the values between 0 and 70 degrees inclusive.

Statements

If-statement

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

There is no "else if" statement; use a trick to get the same effect. Combine the "if" and "else" on one line, and end with an "end if" for each “if” statement:

```
  if(scalar_logical_expression_A) then
     [statement(s)]
  else if(scalar_logical_expression_B) then
     [statement(s)]
  else if(scalar_logical_expression_C) then
     [statement(s)]
  else
     [statement(s)]
  end if  ; expression C (includes the "else")
  end if  ; expression B
  end if  ; expression A
```

Loops

Loops are useful but may not be efficient; they should be used minimally. Use array arithmetic and/or built-in functions if available.

```
   do n=start,end[,stride]
       [statement(s)]
   end do  ; the stride is not optional if end < start
```

Loop while a logical expression is True:

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

Use “continue” to skip to next loop iteration; “break” to exit a loop.

Assignment/Reassignment

Assign a variable:

```
   var = "This is a string"      ; type string
```

Reassign the variable with a different type and shape:

```
   var := (/1,2,3,4/)            ; type integer
```

Metadata and attributes

Metadata is the information associated with a variable or file that describes the data. The metadata of a variable can be attributes like *units*, *_FillValue*, and for a file it can be *creation_date* and *history*.

```
  var@units      = "degK"
  var@long_name  = "Near Surface Temperature"
  var@_FillValue = -99999

  title = var@long_name
```

Get the attributes of a variable "slp" of a file "file_name.nc":

```
  fin       = addfile("file_name.nc","r")
  file_atts = getfilevaratts(fin,"slp")
```

To verify whether an attribute of a variable exists, use *isatt*:

```
   if(isatt(slp,"units")) then
      print(slp@units)
   end if
```

Print

Print procedures echoing to stdout (standard out).

1. Prints all the values of a variable or expression
```
        print(variable_or_expression or file)
```
2. Prints summary of a variable's information (commonly used)
```
        printVarSummary(data_variable)
```
3. Formatted print of all elements from a list
```
        print_table(list)
```
4. Prints the minimum and maximum value of a variable
```
        printMinMax(data_variable,0)
```
5. Prints a summary of a file variable's information
```
        printFileVarSummary(file,varname)
```

Free memory

Use the **delete** procedure to free memory. It can be used to delete a single variable or a variable list.

```
delete(var)
delete([/var1,var2,var3/])
```

User-defined functions and procedures

Generally, functions return values; procedures perform tasks. They must have a **begin** and an **end** statement.

Procedures:

```
  undef("procedure_name")
  procedure procedure_name(declaration_list)
  local local_variables  ; optional, but recommended
  begin
     statements
  end
```

Functions:

```
  undef("function_name")
  function function_name(declaration_list)
  local local_variables  ; optional, but recommended
  begin
     statements
     return(return_variable)
  end
```

Functions can return multiple variables contained within a variable of type list:

```
  undef("ret_mulvar")
  function ret_mulvar(val1,val2)
  local ni,nj     ; optional, but recommended
  begin
     ni = val1 + val2
     nj = val1 * val2
     return([/ni,nj/])      ; return value list
  end

    comp  = ret_mulvar(5,2) ; call function
    v_add = comp[0]        ; retrieve 1st list element
    v_mul = comp[1]        ; retrieve 2nd list element
```

Important built-in functions and procedures

all / any	Returns True if all/any of the values of its input evaluate as True
cd_calendar	Converts a mixed Julian/Gregorian date t a UT-referenced date
conform	Conforms an array to the shape of anothe
dimsizes	Returns dimension sizes of input variabl
exit	Forces an NCL script to exit immediately
ind	Returns indices where the input is True
ismissing	Returns True for every element of the input that contains a missing value
num	Counts the number of True values in inpu
systemfunc	Executes shell command and returns outpu
typeof	Returns type of input variable
where	Performs array assignments based on a conditional array