NCAR COMMAND LANGUAGE (NCL)

Objective Climate Forecasts for Agriculture and Food Security Sector in Eastern and Southern Africa Training of Trainers Workshop

31st August
Victoria Falls, Zimbabwe
INTRODUCTION

• NCL (NCAR Command Language) is an interpreted language designed for access, analysis and visualization of data.

• The software is open source and was developed with emphasis on Atmospheric Science data.

• A number of input files are supported by NCL these include NetCDF, GRIB1, GRIB2, HDF4, HDF5, Binary, Shapefiles and Ascii.

• NCL can be run in both interactive mode (command prompt ) and batch process using scripts.
RESERVED KEY WORDS

• NCL has several reserved keywords, if you try to use these key words for your variables you will likely get errors. In addition to the keywords all build in functions and procedure names are reserved.

begin, break, byte, character, continue, create, do, double, else, end, numeric, external, False, file, float, function, getvalues, graphic, if, integer, int64, list, load, local, Logical, long, quit, record, setvalues, short, string, then, undef, while, True
# SYNTAX CHARACTERS

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Assignment syntax</td>
</tr>
<tr>
<td>;</td>
<td>Starts a comment</td>
</tr>
<tr>
<td>-&gt;</td>
<td>For inputting/outputting supported data formats</td>
</tr>
<tr>
<td>(/...../)</td>
<td>Array constructor</td>
</tr>
<tr>
<td>{.......}</td>
<td>Subscript array using coordinate values</td>
</tr>
<tr>
<td>\</td>
<td>Continuation character for wrapping long lines</td>
</tr>
<tr>
<td>@</td>
<td>create or reference an attribute</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>create or reference a coordinate variable</td>
</tr>
<tr>
<td>::</td>
<td>Separator when calling external codes</td>
</tr>
<tr>
<td>$ $</td>
<td>Enclose strings when importing or exporting variables</td>
</tr>
<tr>
<td>!</td>
<td>Create or reference a named dimension</td>
</tr>
</tbody>
</table>
READING IN FILES

- Two main functions are available for reading in files (NetCDF (.nc), Grib1 (.grb), Grib2 (.grb2), HDF4 (.hdf), HDF5 (.hdf), and Shapefiles (.shp)) `addfile` and `addfiles`. The function `addfile` can also be used to create and write files.

```
in = addfile("PrecCPTcanCor_3monthSeasonal.nc","r")
```

- **Function used to read in files**
- **Status, in this case its read (r). One can also create (c) or write (w) a file**
- **Reference to data file**
- **File to be read in**
VARIABLES IN A FILE

• A variable is an element in a file, which can change. The variables are **case sensitive** thus *RAIN* and *rain* are different variables.

• Variable names must begin with an alphabetic character and can contain numeric characters.

• In most cases when the variable is read it also contains its metadata.

• To have an idea of the variables that are in a file, there are 2 functions that can be utilized using `ncdump` or `ncl Filedump`.
**READ IN A VARIABLE**

- To read a variable from any supported file format with all metadata information included:
  
  ```
  fin = addfile("PrecCPTcanCor_3monthSeasonal.nc", "r")
  t = fin->corr
  ```

- To strip off the metadata, enclose the file variable reference with ‘(/**/’). Only, the special _FillValue attribute will be carried over.
  
  ```
  fin = addfile("PrecCPTcanCor_3monthSeasonal.nc", "r")
  t = (/ fin->corr /)
  ```
There are cases when one might want to access multiple data files. In this case the function addfiles is utilized in conjunction with `systemfunc`. The function systemfunc execute a shell command and returns the output.

```python
files = systemfunc("ls wrfdaily*.nc")
in=addfiles(files, "r")
```

files: list of reference to the multiple data files
Two options are available for importing a variable into memory, the *cat* and *join*. These can be achieved using the *ListSetType* command. The default option is *cat*.

\[
\text{T2Mean}=\text{in}[:]\rightarrow\text{T2MEAN}
\]

Since *cat* is the default option, to use the *join* option one needs to specify this through using *ListSetType*

\[
\text{ListSetType}(\text{in}, \text{“join”})
\]

\[
\text{T2Mean}=\text{in}[:]\rightarrow\text{T2MEAN}
\]
ARRAYS

• Array operations require that all arrays conform to each other. This means that the arrays must have the same size and shape.

• NCL also automatically handles missing values.

• There are three types of subscripting:
  – Standard
  – Named
  – Coordinate

• Remember, the subscription of an array or dimension starts, like in C, with the index 0.

  e.g. \( a = (/ 4, 2, 1, 3 /) \) 4 elements; index 0-3
The subscripts used in standard subscripting are integers.

The most general form of a standard subscript is $x:y:i$ which indicates the range $x$ to $y$ in strides of $i$.

Consider the array $a$ defined by
- $a = (5,6,7,8,9, 10,1, 2,4,2)$
- $b = a(0:4:2)$ subset the first 5 and time-steps of 2: this should give an array containing $(5,7,9)$
- $c = a(:4)$ subset first 4 with time-step of 1
- $d = a(:,:,1)$ flips the order
ARRAYS: NAMED SUBSCRIPTING

- Named subscripting allows you to reorder arrays, but is only allowed when all dimensions of the array are named dimensions.

- Let us use a variable `precip` that has two dimensions named "lon" and "lat". The dimension "lat" is of size 61 and the dimension "lon" is of size 56:

  Re-order the dimensions:
  
  ```
  p_reord1 = precip(lon|:, lat|:)
  p_reord2 = precip(lon|19:39, lat|0:9)
  ```
ARRAYS: COORDINATE SUBSCRIPTING

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

• Example array
  \[ m = ( -5.0, 10.0, 15.0, 20.0, 25.0, 30.0 ) \]
  \[ m!0 = \text{"lat"} \text{ – name the dimension} \]
  \[ m&lat = m \text{ – associate the array} \]
  \[ mw = m(\{-5.0 : 25.0 : 2\}) \text{ contains the values – 5.0, 15.0, 25.0} \]
• In this section we learn how to plot the data. NCL can write the following graphics: ps, eps, pdf, png, and X11.

• X11 is good when the script is being tested and is only able to display the plots but does not save the output.

• There are 5 major parts that are required for plotting
  – Load the functions or procedures
  – Open data file
  – Define the variable
  – Define the plot resources
  – Plot
load "${NCARG_ROOT}/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "${NCARG_ROOT}/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
  y = cos(0.2*isp(0,50,1)) ; 51 points

  wks = gsn_open_wks("ps","test") ; `test.ps`
  res = True ; plot options
  res@xyLineColor = "Blue" ; line color
  res@tiMainString = "Practise plot"
  res@gsnMaximize = True

  plot = gsn_csm_y(wks,y,res) ; no X values
end
SHAPEFILES

• A shapefile is geospatial vector data format for GIS system software. Mostly used to mask data to a specified region. Shapefiles can have three different types of data
  – Point (locations of cities or places or interest, election data)
  – Polyline (non-closed boundaries like rivers and roads)
  – Polygon (closed geographic boundaries like countries)

• Only one data type per shapefile.

• NB: Shapefiles are read in the same way with other files that is use addfile however you need the .shp, .shx and .dbf in the same directory.
USING SHAPEFILES WHEN PLOTTING

Extract from plotting script

```r
f = addfile(shpName ,"r")
state_lon = f->x
state_lat = f->y

cntr = gsn_csm_contour_map(wks,xvar,optsr)
poly1 = gsn_add_shapefile_polylines(wks,cntr,shpName,lnres)
draw(cntr)
frame(wks)
```

Read in the shapefile

Function to addshapefile to a plot
NCL is an interpreted language designed specifically for scientific data analysis and visualization.

Portable, robust, and free, NCL is available as binaries or open source.

Supports NetCDF 3/4, GRIB 1/2, HDF 4/5, HDF-EOS 2/5, shapefile, ASCII, binary.

Numerous analysis functions are built-in.

High-quality graphics are easily created and customized with hundreds of graphic resources.

Many example scripts and their corresponding graphics are available.

nci.ucar.edu
THANK YOU