PYTHON APLICADO A LA CLIMATOLOGÍA

Bch. Javier Chiong Ravina

NetCDF

https://unidata.github.io/netcdf4-python

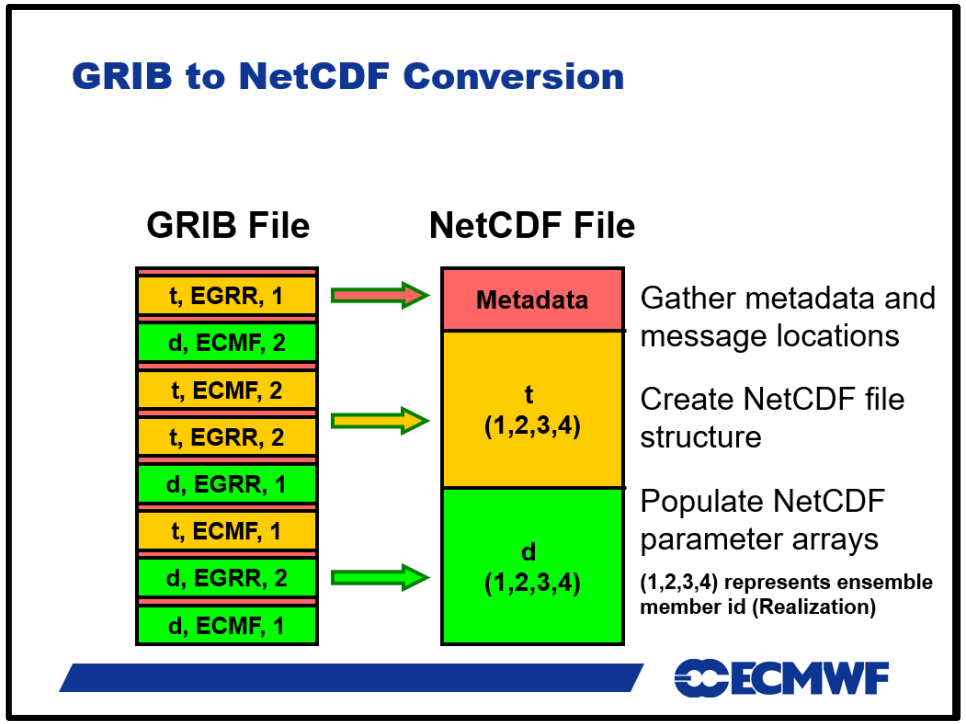
Desarrollado por UNIDATA

Esta libreria puede leer y escribir archivos tanto en el nuevo formato netCDF 4 como en el antiguo netCDF 3.

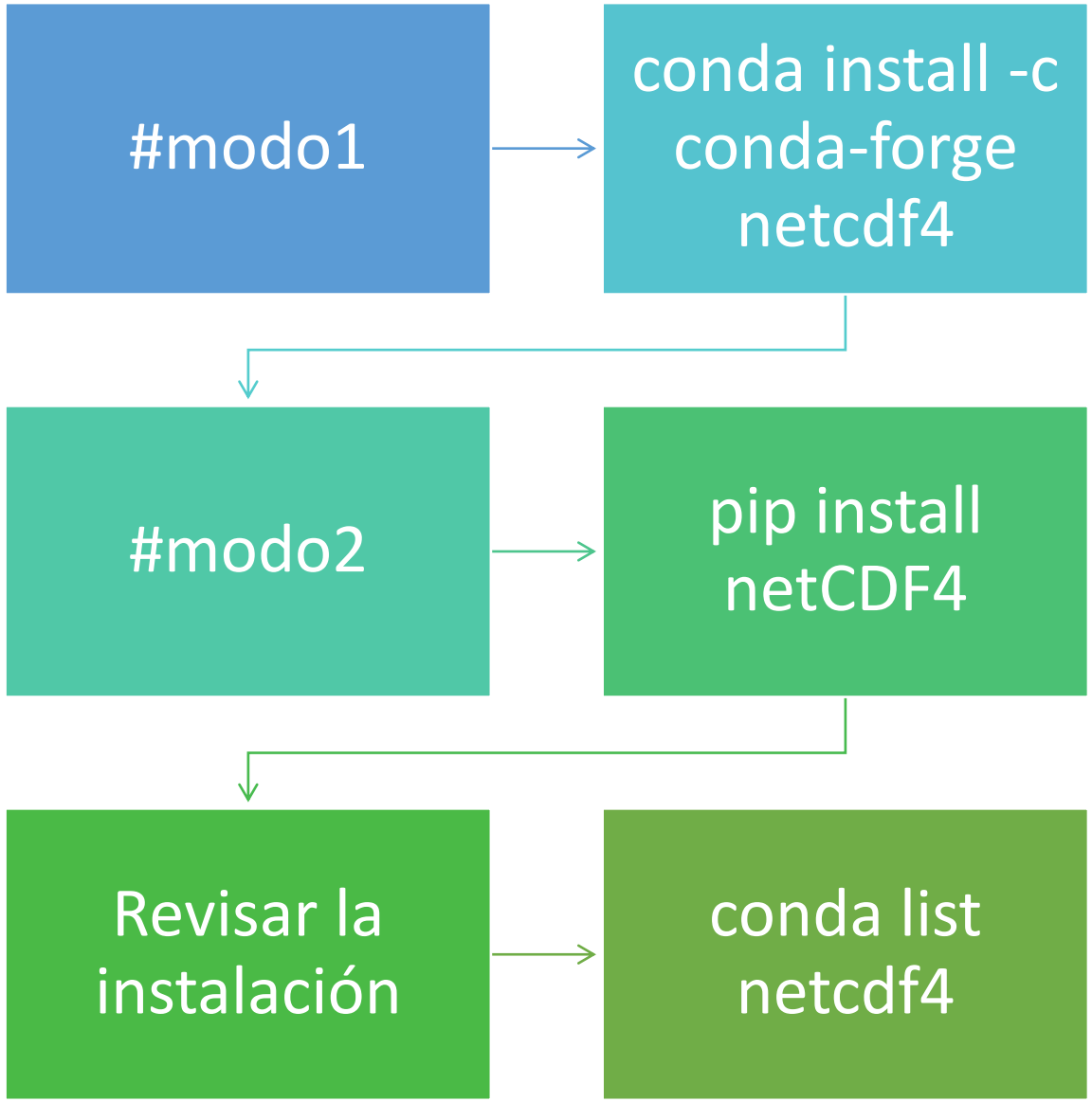
Archivos formato NetCDF:

Network Common Data Form (NetCDF) se utiliza habitualmente para almacenar datos geográficos multidimensionales. Es un conjunto de bibliotecas de software y formatos de datos independientes de la máquina que permiten crear, acceder y compartir datos científicos orientados a matrices. NetCDF4 esta basado en HDF5.

Algunos ejemplos de estos datos son la temperatura, la precipitación y la velocidad del viento. Las variables almacenadas en NetCDF suelen medirse varias veces al día en grandes áreas (continentales). Con múltiples mediciones por día, los valores de los datos se acumulan rápidamente y se vuelven difíciles de manejar.



Instalación



Procesamiento de archivos netCDF

El procesamiento de los archivos netcdf es realizada con la librería **netCDF4** de python

Para abrir archivos netcdf utilizamos utilizamos la siguiente función:

```
Dataset(name, mode='r', format='NETCDF4')
```

donde:

name : indica el nombre del archivo netcdf

mode : indica la acción que se hará con el archivo. Para leer se utiliza 'r' y para escribir 'w'.

format : indica la versión del netcdf.

Para importar esta función utilizamos la siguiente sintaxis:

```
from netCDF4 import Dataset
```

Exploración de atributos

```
In [2]: ds
Out[2]:
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF3_64BIT_OFFSET data model, file format NETCDF3):
    Conventions: CF-1.6
    history: 2021-11-26 00:20:26 GMT by grib_to_netcdf-2.23.0: /opt/ecmwf/mars-client/bin/
grib_to_netcdf -S param -o /cache/data8/adaptor.mars.internal-1637886025.710311-4024-11-
f53bbad8-5b23-4bbb-a2b4-d6b1da9f6ac4.nc /cache/tmp/f53bbad8-5b23-4bbb-a2b4-d6b1da9f6ac4-
adaptor.mars.internal-1637886020.741077-4024-18-tmp.grib
    dimensions(sizes): longitude(241), latitude(301), level(5), time(4)
    variables(dimensions): float32 longitude(longitude), float32 latitude(latitude), int32
level(level), int32 time(time), int16 r(time, level, latitude, longitude), int16 t(time, level,
latitude, longitude), int16 u(time, level, latitude, longitude), int16 v(time, level, latitude,
longitude)
    groups:
```

ATRIBUTOS

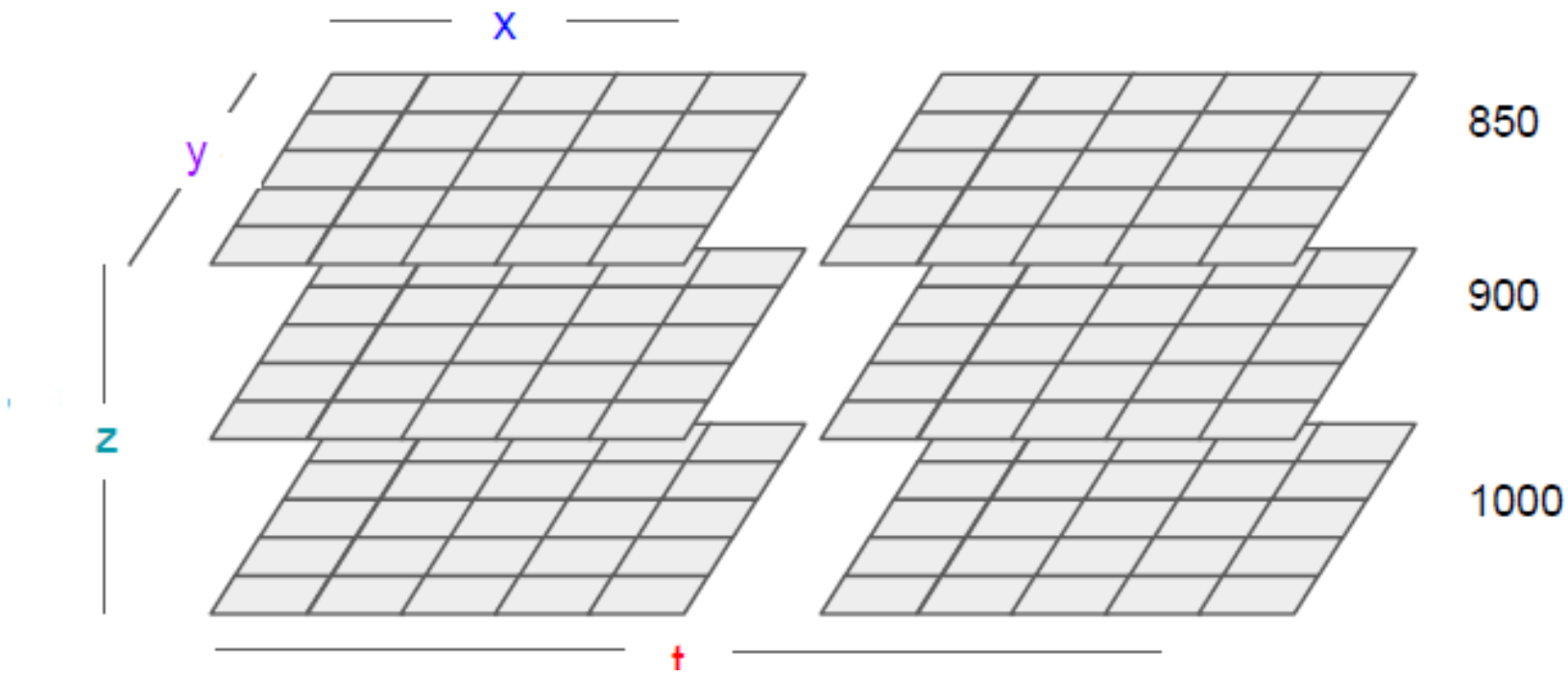
DIMENSIONES

VARIABLES

GRUPOS

Ejercicio 1

```
#Accedemos a las distintas partes del archivo
ds
#Parte de los atributos del archivo, pueden variar ds.Conventions
#Parte de los atributos del archivo, pueden variar ds.history
#Obtener las dimensiones del archivo
ds.dimensions
#Obtener los grupos del archivo ds.groups
#Acceder a cada dimension del archivo por su etiqueta
ds.dimensions['longitude']
ds.dimensions['latitude']
ds.dimensions['level']
ds.dimensions['time']
#Acceder a las variables del archivo
ds.variables['r']
ds.variables['t']
ds.variables['u']
ds.variables['v']
ds.variables['longitude']
ds.variables['latitude']
ds.variables['level']
ds.variables['time']
```



Estructura de datos de un archivo netcdf

- Dimensiones:
- X, Y, Z asociados a la longitud, latitud y niveles isobaricos.
- Variables: A manera de ejemplo las variables pueden ser Temperatura, Hr, viento.

Obtención de datos

```
In [2]: ds
Out[2]:
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF3_64BIT_OFFSET data model, file format NETCDF3):
    Conventions: CF-1.6
    history: 2021-11-26 00:20:26 GMT by grib_to_netcdf-2.23.0: /opt/ecmwf/mars-client/bin/
grib_to_netcdf -S param -o /cache/data8/adaptor.mars.internal-1637886025.710311-4024-11-
f53bbad8-5b23-4bbb-a2b4-d6b1da9f6ac4.nc /cache/tmp/f53bbad8-5b23-4bbb-a2b4-d6b1da9f6ac4-
adaptor.mars.internal-1637886020.741077-4024-18-tmp.grib
    dimensions(sizes): longitude(241), latitude(301), level(5), time(4)
    variables(dimensions): float32 longitude(longitude), float32 latitude(latitude), int32
level(level), int32 time(time), int16 r(time, level, latitude, longitude), int16 t(time, level,
latitude, longitude), int16 u(time, level, latitude, longitude), int16 v(time, level, latitude,
longitude)
    groups:
```

VARIABLES

Ejercicio 2

```
#Accedemos y guardamos los datos de las variables del archivo netcdf
hr=ds.variables['r'][:].data
temp=ds.variables['t'][:].data - 273.15
u=ds.variables['u'][:].data
v=ds.variables['v'][:].data
#Tambien las dimensiones temporales y espaciales estaran expresadas como variables
longitude=ds.variables['longitude'][:].data
latitude=ds.variables['latitude'][:].data
level=ds.variables['level'][:].data
```

Ejercicio 3

Conversión de fechas de formato numérico a calendario

```
In [5]: print(ds.variables['time'][:].data)
[1052592 1052598 1052604 1052610]
```

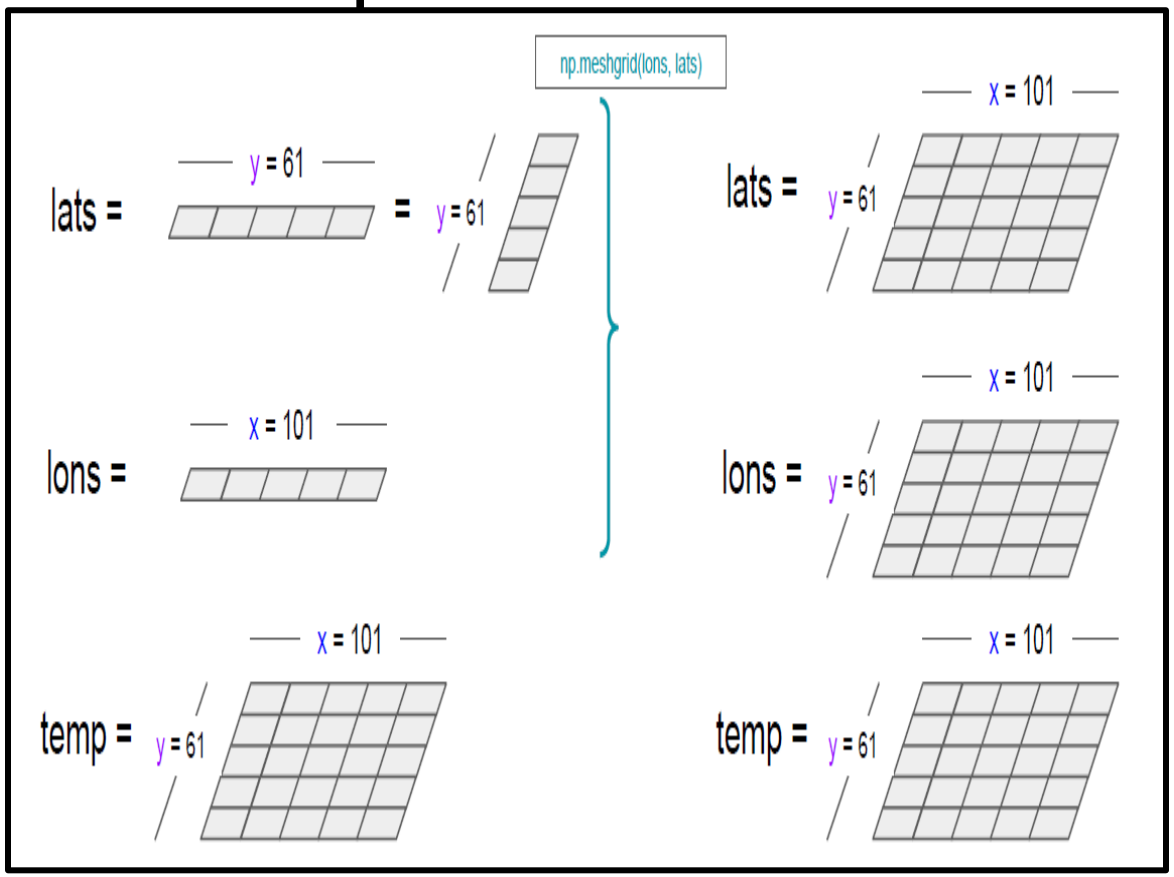
FORMATO NUMERICO

```
In [6]: date_time
Out[6]:
array([datetime.datetime(2020, 1, 30, 0, 0),
       datetime.datetime(2020, 1, 30, 6, 0),
       datetime.datetime(2020, 1, 30, 12, 0),
       datetime.datetime(2020, 1, 30, 18, 0)], dtype=object)
```

FORMATO YY,MM,DD,HH

```
print(ds.variables['time'])
print(ds.variables['time'][:].data)

date_time_num = ds.variables['time'][:].data

units = ds.variables['time'].units

calendar = ds.variables['time'].calendar

date_time = num2date(date_time_num, units=units,
calendar=calendar,
only_use_cftime_datetimes=False,
only_use_python_datetimes=True)
```

Selección de niveles y tiempos



Ejercicio 4

```
#Obtencion de la variable temperatura en un tiempo y nivel determinado
temp850_t1 = temp[0,2,:,:]
temp925_t1 = temp[0,3,:,:]
#Obtencion de la variable velocidad zonal y meridional en un tiempo y nivel determinado
u850_t1 = u[0,2,:,:]
v850_t1 = v[0,2,:,:]
u925_t1 = u[0,3,:,:]
v925_t1 = v[0,3,:,:]
#Calculo de la rapidez
vel850_t1 = np.sqrt(u850_t1**2 + v850_t1**2)
vel925_t1 = np.sqrt(u925_t1**2 + v925_t1**2)
#Transformación de valores de latitud y longitud de 1D a 2D
lons2D, lats2D = np.meshgrid(longitude, latitude)
```

Pygrib

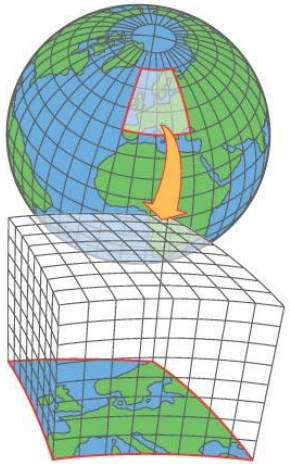
https://pypi.org/project/pygrib/

Desarrollado por Jeff Whitaker (NOAA). Viene a ser una interfaz del paquete ECCODES(https://confluence.ecmwf.int/display/ECC) desarrollado por ECMWF, éste paquete ofrece un conjunto de herramientas para codificar y decodificar los siguientes formatos:

- WMO FM-92 GRIB edición 1 y edición 2
- WMO FM-94 BUFR edicion 3 y edicion 4
- WMO GTS encabezado abreviado (solo decodifica).

Los archivos GRidded Information in Binary (GRIB) son formatos para el almacenamiento y el transporte de datos meteorológicos grillados. Está diseñado para ser autodescriptivo, compacto y portátil entre arquitecturas informáticas. El estándar GRIB fue diseñado y es mantenido por la Organización Meteorológica Mundial (OMM).

- GRIB Edición 0: actualmente obsoleta, sin soporte y raramente utilizada.
- GRIB Edición 1: ya no es la edición GRIB más actual de la OMM, este formato ha sido congelado de cara a futuras mejoras. Sin embargo, debido a su uso en el sistema de previsión de OACI, sigue siendo reconocido por la OMM.
- GRIB Edición 2(GRIB2): representa una ampliación y una modernización significativa de la norma GRIB

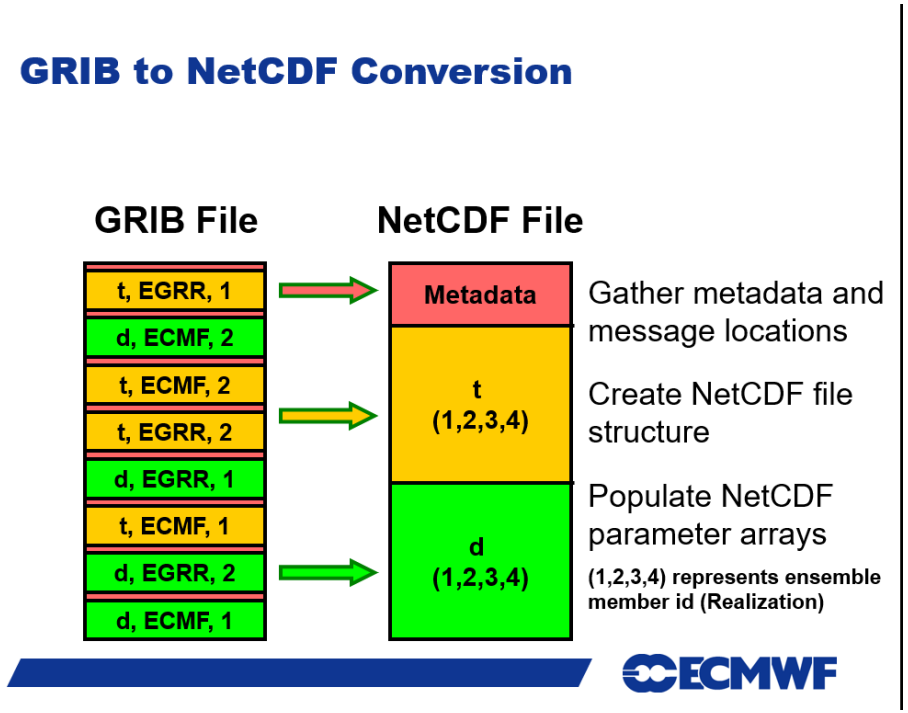


Fuente:https://www.csc.fi/fi/artikkelit

Fuente: cosmo.model.org

¿Qué contiene un archivo GRIB?

Un archivo GRIB contiene uno o más registros de datos, organizados como un flujo de bits secuencial. Cada registro comienza con una cabecera, seguida de datos binarios empaquetados. La cabecera está compuesta por números de 8 bits sin signo (octetos). Contiene información sobre la naturaleza cualitativa de los datos (campo, nivel, fecha de producción, tiempo de validez previsto, etc.) la propia cabecera (metainformación sobre la longitud de la cabecera, el uso de los bytes de la cabecera, la presencia de subcabeceras opcionales) el método y los parámetros que se utilizarán para descodificar los datos empaquetados la disposición y las características geográficas de la cuadrícula en la que se van a trazar los datos.



Fuente: Baudouin Raoult Data and Services Section ECMWF

Instalación

- conda install pygrib **problemas de instalación**
- **conda install -c conda-forge pygrib recomendado**
- pip install pygrib **problemas de instalación**
- conda list pygrib

Revisar el paquete y sus dependencias:

- conda search pygrib --info

Procesamiento de archivos grib y grib2

El procesamiento de los archivos grib y grib2 es realizada con la librería **pygrib** de python (https://jswhit.github.io/pygrib/).

Para abrir archivos grib utilizamos utilizamos la siguiente función:

- import **pygrib as pg**
- pygrib.open(nombre_archivo)

Métodos:

- .read() : lista el contenido del archivo
- .select(name=nombre_var, typeOfLevel=tipo_nivel, level=nivel) : selecciona una variable variable en función de su nombre, el tipo de su nivel y su nivel
- .data(lat1=latmin, lat2=latmax, lon1=lonmin, lon2=lonmax) : selecciona una región del array de la variable, retorna la variable, latitud (2d) y longitud (2d)
- .latlons() : retorna la latitud (2d) y la longitud (2d)
- .keys() : lista todos los parámetros asociados a una variable
- .latitudes : retorna la latitud (1d)
- .longitudes : retorna la longitud (1d)
- .level : retorna el nivel
- .values : retorna el array de la variable
- .dataDate : retorna la fecha
- .dataTime : retorna la hora
- .forecastTime : retorna las horas posteriores al tiempo inicial del dato
- .validityDate : retorna la fecha para la cual es valida la variable
- .validityTime : retorna la hora para la cual es valida la variable

Ejercicio1

```
#importar lib
import numpy as np
import pygrib as pg
grbs = pg.open('C:/Users/drago/Desktop/CURSO_PY/era5-levels-members.grib')
 #revisamos las variables
for grb in grbs:
        print(grb)
```

Exploración de atributos

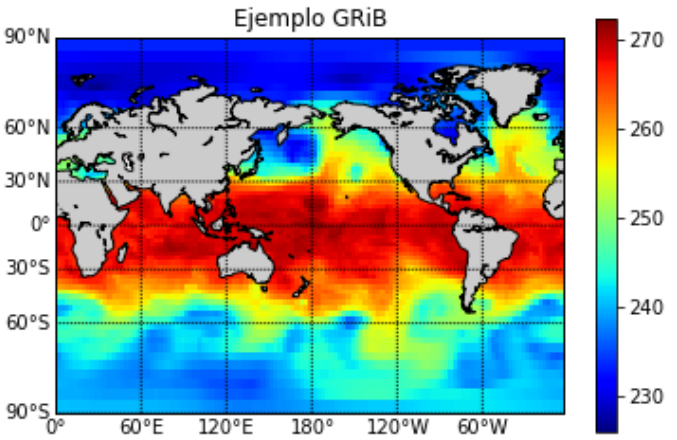
Ejercicio 1 (continuacion)

```
#de la lista de variables seleccionar Temperature #muestra el primero de la lista
grb = grbs.select(name='Temperature')[0]
grb
#extraer los valores de los datos
data = grb.values
data
#explorar el array data
data.shape, data.min(), data.max()
#seleccionar una variable por numeracion
grb2=grbs.message(160)
grb2
g = grbs[160]
g.messagenumber
#explorar atributos
g.keys()
#de la lista de atributos
 #exploramos los valores de la variable 160
 g['values']
#determinamos los missing value
g['missingValue']
#determinamos el nivel
g['level']
```

Plot básico

```
#selecionamos una variable
grb = grbs.select(name='Temperature')[0]
data= grb.values
data
lats, lons=grb.latlons() lats.shape, lats.min(), lats.max(), lons.shape, lons.min(), lons.max()
m=Basemap(projection='mill', lat_ts=10, llcrnrlon=lons.min(), urcrnrlon=lons.max(), llcrnrlat=lats.min(),
urcrnrlat= lats.max(), resolution='c')
x, y = m(lons,lats)
cs = m.pcolormesh(x, y, data, shading='flat', cmap=plt.cm.jet)
m.drawcoastlines()
m.fillcontinents()
m.drawmapboundary()
m.drawparallels(np.arange(-90.,120.,30.),labels=[1,0,0,0])
m.drawmeridians(np.arange(-180.,180.,60.),labels=[0,0,0,1])
plt.colorbar(cs,orientation='vertical')
plt.title('Ejemplo GRiB') plt.show()
```

Ejercicio 1 (continuación)



h5py

https://www.h5py.org/

Desarrollado por h5py Project para manipular datos con formato HDF5

Puede dividir Datasets como arrays de Numpy

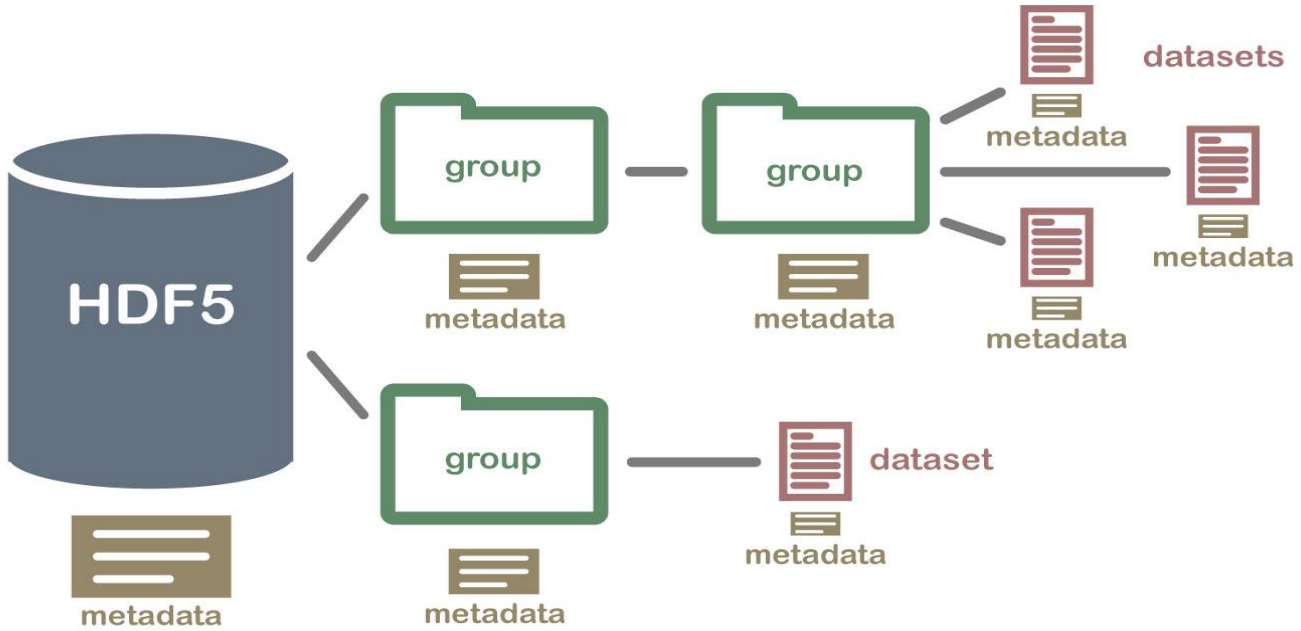
Lo más fundamental que hay que recordar al usar h5py es:

Los Grupos funcionan como diccionarios, y los Dataset funcionan como arrays de NumPy.

Hieralchical Data Format Version 5 (HDF5) es un formato de archivo de código abierto que soporta datos grandes, complejos y heterogéneos. HDF5 utiliza una estructura parecida a la de un "directorio de archivos" que permite organizar los datos dentro del archivo de muchas formas estructuradas diferentes, como lo podría hacer con los archivos de su ordenador. El formato HDF5 también permite incrustar metadatos que lo hacen autodescriptivo.

Un archivo HDF5 es un contenedor de dos tipos de objetos: los **Dataset**, que son colecciones de datos tipo array, y los **Grupos**, que son contenedores tipo carpeta que contienen datasets y otros grupos.

Estructura de un archivo HDF5



Fuente: https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5

Árbol de dependencias de un archivo HDF5 contiene (similar a un directorio de una computadora)

- Grupos: Carpetas que contienen subgrupos y datasets.
- Datasets: Son arrays numpy conteniendo la data.
- Posee metada.

Instalación

conda install –c conda-forge h5py

conda install h5py

pip install h5py

Revisar el paquete y sus dependencias:

conda search h5py --info

HDF5 fue diseñado específicamente:

- Para datos de gran volumen y/o complejos (pero puede utilizarse para datos de bajo volumen/simples)
- Para cualquier tamaño y tipo de sistema (portátil)
- Para un almacenamiento y entrada/salida (I/O) flexibles y eficientes
- Para permitir que las aplicaciones evolucionen en su uso de HDF5 y para dar cabida a nuevos modelos
- Para ser utilizado como un kit de herramientas de formato de archivo (muchos otros formatos utilizan HDF5 como base Ej NetCDF)

Ejercicio 2

```
import h5py
ar = h5py.File('C:/Users/drago/Desktop/CURSO_PY/NEONDSTowerTemperatureData.hdf5', 'r’)
 #revisar el contenido del archivo hdf
#forma1
list(ar.keys())
#forma2
for key in ar.keys():
  print(key)
```

Ejercicio 2 (continuación)

```
print(type(ar)) #revisar que clase es
#es un archivo que contiene dos Grupos=['Domain_03', 'Domain_10’]
/elegimos un grupo y seleccionamos
grp = ar['Domain_10’]
grp
#revisamos que clase es
print(type(grp)) #la clase es Grupo
#revisamos el contenido del Grupo=grp
list(grp.keys())
# es un grupo que contiene otro grupo ['STER'] #continuamos explorando
grp2 = grp['STER’]
list(grp2.keys()) #contiene dos Grupos = ['min_1', 'min_30’]
grp3 = grp2['min_1’]
list(grp3.keys()) #contiene 3 Grupos
 grp4 = grp3['boom_1’]
list(grp4.keys())
grp4
```

Ejercicio 2 (continuación)

```
#extarer el path del grupo
grp4.name
# contiene el Grupo o Dataset ['temperature’]
dset1 = grp4['temperature’]
print(type(dset1)) # determinamos que es un Dataset
dset1
 #revisamos contenido
dset1[:]
dset1[0]
```

Ejercicio 2 (continuación)

Para hallar el árbol de dependencias se puede simplificar con el siguiente código:

```
#explorar el tipo de contenido
[x for x in ar]
for name in ar:
print(name,ar.get(name, getclass=True))
#Determinar todos los grupos, subgrupos y datasets
#(arbol de dependencias)
def printname(name):
print(name)
# visualizacion de arbol de dependencias
ar.visit(printname)
```

Ejercicio 3: Plot básico

```
import os
import h5py
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
FILE_NAME= 'C:/Users/drago/Desktop/CURSO_PY/1B.GPM.GMI.TB2016.20160105-S230545-E003816.010538.V05A.hdf5’
with h5py.File(FILE_NAME, mode='r’) as f:
# extraer el dataset Tb usando path
ds = '/S1/Tb’
data = f[ds][:,:,0]
units = f[ds].attrs['units’]
_FillValue = f[ds].attrs['_FillValue'] data = np.ma.masked_where(np.isnan(data), data)
# Datos de ubicacion usando direccion  o path
latitude = f['/S1/Latitude'][:]
longitude = f['/S1/Longitude'][:]
```

Ejercicio 3: Plot básico

```
#graficas
m = Basemap(projection='cyl', resolution='l',
            llcrnrlat=-90, urcrnrlat=90,
            llcrnrlon=-180, urcrnrlon=180)
m.drawcoastlines(linewidth=0.5)
m.drawparallels(np.arange(-90, 91, 45))
m.drawmeridians(np.arange(-180, 180, 45), labels=[True,False,False,True])
m.scatter(longitude, latitude, c=data, s=1, cmap=plt.cm.jet,
          edgecolors=None, linewidth=0)
cb = m.colorbar(location="bottom", pad='10%')
cb.set_label(units)
basename = os.path.basename(FILE_NAME)
plt.title('{0}\n{1}'.format(basename, ds +' (nchan1=0)'))
fig = plt.gcf()
```