Python and Grib API

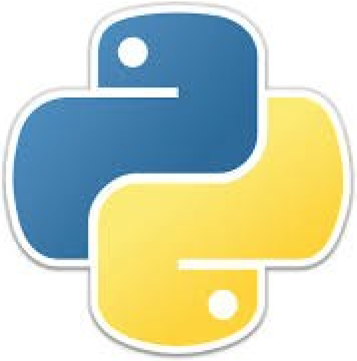
Xavi Abellan
User Support Section

Python and GRIB API

- Just an appetizer
- Provide you only a small view of the world the Python interface opens to
- Increase your awareness
- You need to explore

What is Python?

- Interpreted, high level scripting language
- Strong, but optional, Object Oriented programming support
- Open-source software, which means it's free
- Easy to learn
- Portable
- Dynamic typing
- Support for exception handling
- Good integration with other languages
- Higher productivity
- Alternative to Matlab, IDL, …
- Through extensions supports many scientific data formats, e.g. netcdf, hdf5, grib, etc.



Python basics: hello world

```
#!/usr/bin/env python
import sys

# This is a comment
def say_hello(name):
    print(“Hello "+ name + "!" )

if len(sys.argv) > 1 :
    name = sys.argv[1]
else:
    name = “World"

say_hello(name)
```

```
$> python example.py
hello World!
$> ./example.py Xavi
hello Xavi!
```

- Import the modules you need
- Indentation to define the different blocks:
  - No ; or { } or END
- Function definition with def
- Variable types not explicitly defined
- Dealing with strings is easy…
- Run with python or directly if shebang present and permissions set

Python basics: list and dicts

```
$> python
Python 2.7.3 (default, Apr  5 2013, 09:29:59)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> mylist = ['a','b','c']
>>> print(mylist)
['a', 'b', 'c']
>>> mylist[2:]
['c']
>>> mylist[-1]
'c'
>>> for element in mylist:
...     print(element)
...
a
b
c
>>> for key,value in mydict.item():
...     print(key + ":" + str(value))
...
key3:3
key2:2
key1:1
>>> 'key1' in mydict
True
>>> 'key5' in mydict
False
>>> len(mydict)
3
>>> mydict.keys()
['key3', 'key2', 'key1']
>>> mydict.values()
[3, 2, 1]
```

NumPy

- Fundamental Python package for scientific computing
- Provides support for multidimensional arrays
- Good assortment of routines for fast operations on arrays
- Performance comparable to that of C or Fortran
- A growing number of Python-based mathematical and scientific packages are using NumPy
- At its core is the ndarray object, an n-dimensional array of homogenous data

```
>>> from numpy import *
>>> a = arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.size
15
>>> b = array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> a.sum()
105
>>> a.min()
0
>>> a.max()
14
>>> a.mean()
7.0
>>> b*2
array([12, 14, 16])
>>> b-b
array([0, 0, 0])
>>> b*b
array([36, 49, 64])
```

NumPy

"""It can be hard to know what functions are available in NumPy."""

http://docs.scipy.org/doc/numpy/reference/

- Operations on arrays:
  - Mathematical and logical
  - Shape manipulation
  - Selection
  - I/O
  - Discrete Fourier transforms
  - Basic linear algebra
  - Basic statistical functions
  - Random simulation

matplotlib

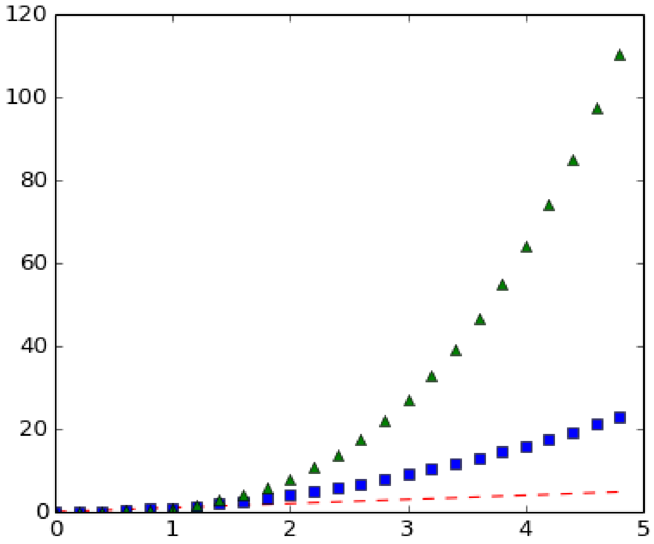
- Plotting library for Python and Numpy extensions
- Has its origins in emulating the MATLAB graphics commands, but it is independent of MATLAB
- Uses NumPy heavily
- Its philosophy is:
  - It should be easy to create plots
  - Plots should look nice
  - Use as few commands as possible to create plots
  - The code used should be easy to understand
  - It should be easy to extend code
- Supports 2D and 3D plotting
- Basemap module: projections, coastlines, political boundaries

matplotlib

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



matplotlib

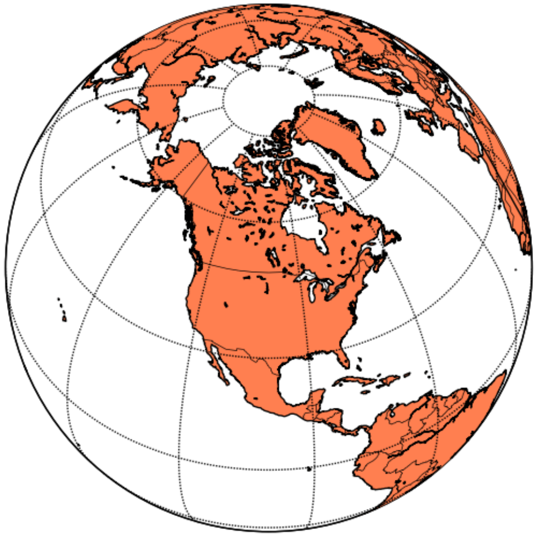
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

# make sure the value of resolution is a lowercase L,
#  for 'low', not a numeral 1
map = Basemap(projection='ortho', lat_0=50, lon_0=-100,
              resolution='l', area_thresh=1000.0)

map.drawcoastlines()
map.drawcountries()
map.fillcontinents(color='coral')
map.drawmapboundary()

map.drawmeridians(np.arange(0, 360, 30))
map.drawparallels(np.arange(-90, 90, 30))

plt.show()
```



SciPy library

- Open source library of scientific algorithms and mathematical tools
- Dependent on NumPy
- Offers improved versions of many NumPy functions
- Quite fast as most of its calculations are implemented in C extension modules
- Offers a decent selection of high level science and engineering modules for:
  - statistics
  - optimization
  - numerical integration
  - linear algebra
  - Fourier transforms
  - signal processing
  - image processing
  - ODE solvers
  - special functions

SciPy Software stack

- Python-based ecosystem of open-source software for mathematics, science, and engineering
- It depends on other python packages like:
  - **Numpy**: Base N-dimensional array package
  - **SciPy library** : Fundamental library for scientific computing
  - **Matplotlib**: Comprehensive 2D Plotting
  - **Ipython**: Enhanced Interactive Console
  - **Sympy**: Symbolic mathematics
  - **Pandas**: Data structures & analysis

Python at ECMWF

- Currently two interfaces for ECMWF libraries
  - GRIB API
  - Magics++
- WREP (Web Re-engineering Project) - ecCharts
- New web plots (GRIB API, magics++)
- Verification (GRIB API, magics++)
- EcFlow (SMS's replacement) - server configuration and client communication
- MACC Project (GRIB API)
- EFAS (European Flood Alert System) (EcFlow)
- Research
- Python interface for future interpolation library is planned

Magics++

- ECMWF's inhouse meteorological plotting software
- Used at ECMWF and in the member states for more than 25 years
- Supports the plotting of contours, wind fields, observations, satellite images, symbols, text, axis and graphs
- Two different ways of plotting
  - Data formats which can be plotted directly: GRIB1, GRIB2, BUFR, ODB, NetCDF and NumPy
  - Data fields can be read with GRIB API, can be modified and then passed to magics++ for plotting
- The produced meteorological plots can be saved in various formats, such as PS, EPS, PDF, GIF, PNG, KML and SVG
- Provides both a procedural and a high-level Python programming interface

Python in GRIB API

- Available since GRIB API version 1.9.5
- Python 2.5 or higher required. Python 3 not yet supported
- Low level, procedural
- Provides almost 1 to 1 mappings to the C API functions
- Uses the NumPy module natively to handle data values
- Should be available by default at ECMWF
- Use module to change the version

Python API – Enabling

- If building the library by hand:

  ./configure --enable-python

- Use of NumPy can be disabled, in which case, Python's native 'array' object will be used. NOT RECOMMENDED

  ./configure –enable-python –disable-numpy

- On 'make install', the Python API related files will go to:

  {prefix}/lib/pythonX.X/site-packages/grib_api

- Either set the PYTHONPATH or link to these files from your Python
- Ready to go: import gribapi

Python interface - Loading/Releasing a GRIB message

gid = grib_new_from_file(file, headers_only=False)

- Returns a handle to a GRIB message in a file
- Requires the input file to be a Python file object
- The use of the headers_only option is not recommended at the moment

gid = grib_new_from_samples(samplename)

- Returns a handle to a message contained in the samples directory

gid = grib_new_from_message(message)

- Returns a handle to a message in memory

grib_release(gid)

- Releases the handle

Python interface - Decoding

value = grib_get(gid, key, type=None)

- Returns the value of the requested key in the message gid is pointing to in its native format. Alternatively, one could choose what format to return the value in (*int*, *str* or *float*) by using the type keyword.

values = grib_get_array(gid, key, type=None)

- Returns the contents of an array key as a NumPy ndarray or Python array. type can only be *int* or *float*.

values = grib_get_values(gid)

- Gets data values as 1D array

values = grib_get_elements(gid, key, indexes)

- Gets a list of particular elements of a given field

On error, a GribInternalError exception (which wraps errors coming from the C API) is thrown.

Python interface - Utilities

[outlat, outlon, value, distance, index] = *grib_find_nearest*(gid, inlat, inlon, is_lsm=False, npoints=1)

- Find the nearest point for a given lat/lon
- (Other possibility is npoints=4 which returns a list of the 4 nearest points)

iter_id = *grib_iterator_new*(gid,mode)

[lat,lon,value] = *grib_iterator_next*(iterid)

grib_iterator_delete(iter_id)

Python interface – Indexing 1/2

iid = *grib_index_new_from_file*(file, keys)

- Returns a handle to the created index
- Release with *grib_index_release*(iid)

grib_index_add_file(iid, file)

- Adds a file to an index.

grib_index_write(iid, file)

- Writes an index to a file for later reuse.

iid = *grib_index_read*(file)

- Loads an index previously saved with *grib_index_write()* to a file.

Python interface – Indexing 2/2

size = *grib_index_get_size*(iid, key)

- Gets the number of distinct values for the index key.

values = *grib_index_get*(iid, key, type=str)

- Gets the distinct values of an index key.

grib_index_select(iid, key, value)

- Selects the message subset with key==value.

gid = *grib_new_from_index*(iid)

- Same as *grib_new_from_file*
- Release with *grib_release*(gid)

Python interface – Encoding

grib_set(gid, key, value)

- Sets the value for a scalar key in a grib message.

grib_set_array(gid, key, value)

- Sets the value for an array key in a grib message.
- The input array can be a numpy.ndarray or a Python sequence like tuple, list, array, ...

grib_set_values(gid, values)

- Utility function to set the contents of the 'values' key.

grib_write(gid, file)

- Writes the message to a file.
- Requires the input file to be a Python file object

Python interface - Cloning

clone_id = *grib_clone*(gid_src)

- Creates a copy of a message.
- You can directly write to file with *grib_write*
- Don’t forget to *grib_release*

Python API – Utilities

values = grib_get_elements(gid, key, indexes)

iter_id = grib_iterator_new(gid,mode)

[lat,lon,value] = grib_iterator_next(iterid)

grib_iterator_delete(iter_id)

Python API – Exception handling

- All GRIB API functions throw the following exception on error: GribInternalError
- Wraps errors coming from the C API

Let's play…

- Go to your SCRATCH and untar the python-handson tarball:

```
$> cd $SCRATCH
$> tar xvzf ~trx/python-grib-practicals.tar.gz
$> cd python-grib-practicals/gribapi
```

- Now, have a look at the grib files with grib_ls

- Run the python interpreter and import the gribapi module:

```
$> python
Python 2.7.3 (default, Apr  5 2013, 09:29:59)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from gribapi import *
>>>
```

Let’s play with sequential access…

- Let’s try some **sequential** access to the grib file 2tmonth.grib:

	HINTS:
1. Open the file	**f = open(‘2tmonth.grib’)**
2. Get the handler for the first message in the file	**grib_new_from_file**
3. Get the value of the ‘dataDate’ and ‘shortName’ parameter	**grib_get**
4. Get the maximum temperature	**max**
5. Release the handler	**grib_release**
6. Now get the handler for the first message in the file and repeat from step 3	**grib_new_from_file**
- Do it for a couple of times at least	
7. Close the file	**f.close()**

Let’s play with indexed access…

Let’s try some **indexed** access to the grib file ztuv.grib:	**HINTS:**
1. Create the index for the file on the dataDate and shortName	**grib_index_new_from_file**
2. Get the different dates and parameters available	**grib_index_get**
3. Select one of the dates and one of the parameters	**grib_index_select**
4. Get the handler for the matching message	**grib_new_from_index**
5. Get the value of the ‘dataDate’ and ‘shortName’ parameter	**grib_get**
6. Release the handler	**grib_release**
7. Now select a different date and parameter and repeat from step 4 ● Do it for a couple of times at least	**grib_index_select**
8. Release the index	**grib_index_release**

Let’s modify a grib message

- Now we are going to create modified version of 2t.grib:

	HINTS:
1. Open the input file	**fin = open(‘2t.grib’)**
2. Get the handler for the first message in the file	**grib_new_from_file**
3. Get the value of the ‘dataDate’ and ‘step’ parameter	**grib_get**
4. Set the dataDate to 20120221 and step to 12	**grib_set**
5. Open the output file for writing	**fout = open(‘2tmod.grib’,’w’)**
6. Write the current modified message to the file	**grib_write**
7. Release the handler	**grib_release**
8. Close the output file	**fout.close()**
9. Close the input file	**fin.close()**
10. Outside the python interpreter, check the new file with grib_ls	

Example scripts

- What is in the directories...
  - gribapi:
    - **index.py:** example on indexed access
    - **reading.py:** example on matplotlib usage
    - **geo.py:** example on iterating over the lat/lon values
  - basemap: example of basemap plotting data from a grib file
    - 2t.py, sst.py
  - magics: example of plotting using Magics++
    - basic_gribapi.py, basic_magics.py colour_gribapi.py magics.py

References

Python specifics:

http://www.python.org/

NumPy

http://numpy.scipy.org/

http://www.scipy.org/Numpy_Functions_by_Category

http://docs.scipy.org/numpy/docs/numpy/

http://www.scipy.org/NumPy_for_Matlab_Users

Langtangen, Hans Petter, ”Python scripting for computational science”

References-cont.

SciPy

http://www.scipy.org/

Matplotlib

http://matplotlib.sourceforge.net/

GRIB API

http://wedit.ecmwf.int/publications/manuals/grib_api/

Questions?