Python and ecCodes

Xavi Abellan

Xavier.Abellan@ecmwf.int

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

- 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

```
#!/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!
```

Python basics: list and dicts

```
$> python
>>> 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

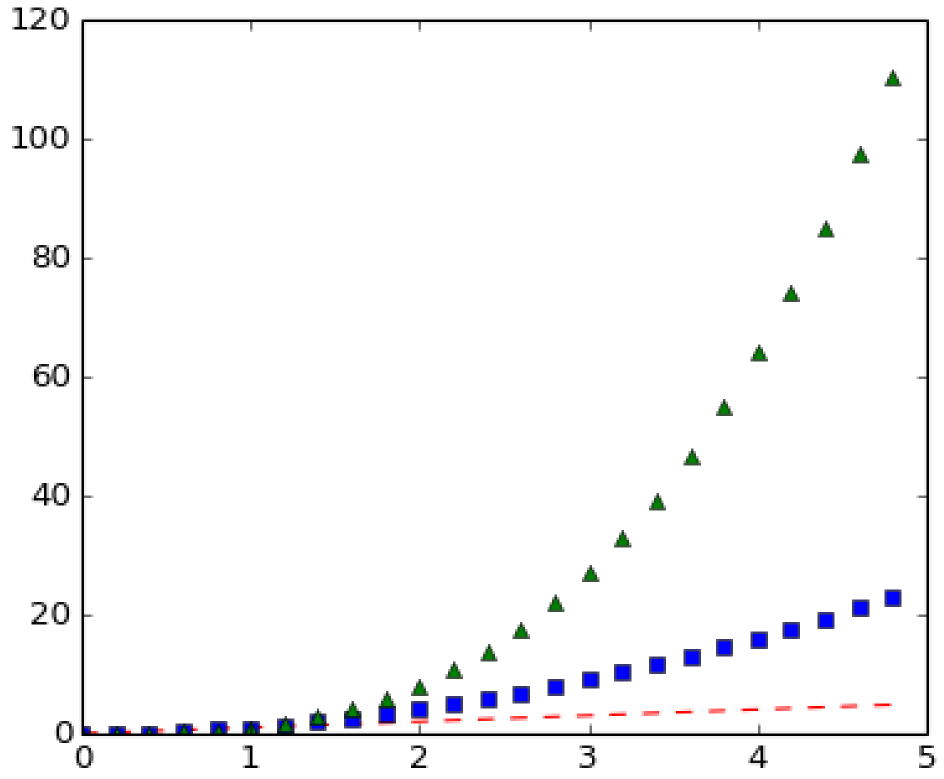
- Plotting library for Python and Numpy extensions
- Has its origins in emulating the MATLAB graphics commands, but it is independent
- Uses NumPy heavily
- Its philosophy is:
  - It should be easy to create plots
  - Plots should look nice
  - Use as few commands as possible to create plots
  - The 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

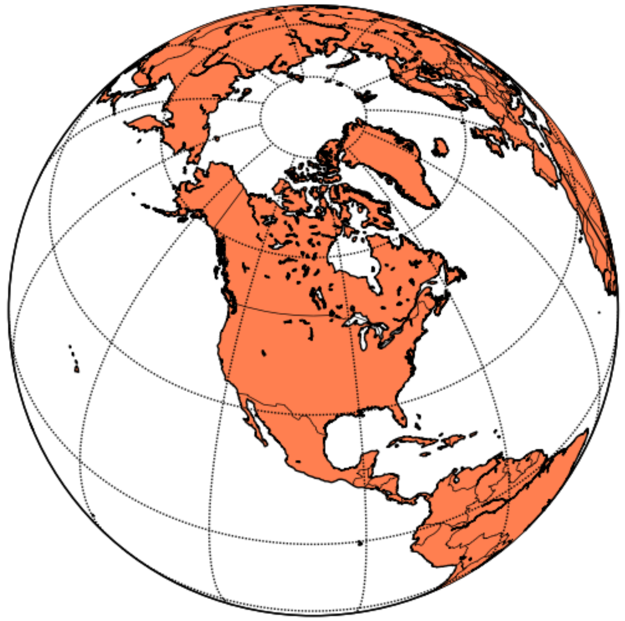
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

# 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
  - ecCodes / GRIB API
  - Magics++
- ecCharts
- New web plots (GRIB API, magics++)
- Verification (GRIB API, magics++)
- EcFlow (SMS's replacement) - server configuration and client communication
- Copernicus 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 ecCodes and 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:

```
cmake –DENABLE_PYTHON ..
```

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

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

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

```
import gribapi
import eccodes
```

Python API – 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.	***codes_grib_new_from_file*** ***codes_any_new_from_file*** ***codes_new_from_file*** (file, **product_kind**, headers_only) - CODES_PRODUCT_GRIB - CODES_PRODUCT_BUFR - CODES_PRODUCT_ANY
gid = ***grib_new_from_samples*** (samplename) Returns a handle to a message contained in the samples directory	***codes_new_from_samples***
gid = ***grib_new_from_message*** (message) Returns a handle to a message in memory	***codes_new_from_message***
grib_release (gid) Releases the handle	***codes_release***

Python API – Decoding

value = ***grib_get*** (gid, key, ktype=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.	***codes_get***
values = ***grib_get_array*** (gid, key, ktype=None) Returns the contents of an array key as a NumPy ndarray or Python array. type can only be int or float.	***codes_get_array***
values = ***grib_get_values*** (gid) Gets data values as 1D array	***codes_get_values***
On error, a ***GribInternalError*** exception (which wraps errors coming from the C API) is thrown	***CodesInternalError***

Python API – 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 With npoints=4 it returns a list of the 4 nearest points	***codes_grib_find_nearest***
iter_id = ***grib_iterator_new*** (gid,mode)	***codes_grib_iterator_new***
[lat,lon,value] = ***grib_iterator_next*** (iterid)	***codes_grib_iterator_next***
grib_iterator_delete (iter_id)	***codes_grib_iterator_delete***

Python API – Indexing

iid = ***grib_index_new_from_file*** (file, keys) Returns a handle to the created index	***codes_index_new_from_file***
grib_index_add_file (iid, file) Adds a file to an index.	***codes_index_add_file***
grib_index_write (iid, file) Writes an index to a file for later reuse.	***codes_index_write***
iid = ***grib_index_read*** (file) Loads an index saved with *grib_index_write* to a file.	***codes_index_read***
grib_index_release (iid) Release the index	***codes_index_release***

Python API – Indexing

size = ***grib_index_get_size*** (iid, key) Gets the number of distinct values for the index key.	***codes_index_get_size***
values = ***grib_index_get*** (iid, key, ktype=str) Gets the distinct values of an index key.	***codes_index_get***
grib_index_select (iid, key, value) Selects the message subset with key==value.	***codes_index_select***
gid = ***grib_new_from_index*** (iid) Same as *grib_new_from_file* Release with *grib_release*(gid)	***codes_new_from_index***

Python API – Encoding

grib_set (gid, key, value) Sets the value for a scalar key in a grib message.	***codes_set***
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, ...	***codes_set_array***
grib_set_values (gid, values) Utility function to set the contents of the 'values' key.	***codes_set_values***
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*	***codes_clone***

Python API – Exception handling

- All GRIB API functions throw the following exception on error:

GribInternalError

- All ecCodes functions throw the following exception on error:

CodesInternalError

- Wraps errors coming from the C API

Questions so far?

Let’s play…

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

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

- Now, have a look at the grib files with grib_ls
- Run the python / Ipython interpreter and import the gribapi / eccodes module

```
> ipython
Python 2.7.3 (default, Apr  5 2013, 09:29:59)
...

In [1]: from gribapi import *

In [2]:
```

```
> ipython
Python 2.7.3 (default, Apr  5 2013, 09:29:59)
...

In [1]: from eccodes import *

In [2]:
```

Let’s play with sequential access…

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

1. Open the file
2. Get the handler for the first message in the file
3. Get the value of the ‘dataDate’ and ‘shortName’ parameter
4. Get the maximum temperature
5. Release the handler
6. Now get the handler for the first message and repeat from step 3
   - Do it for a couple of times at least
7. Close the file

```
HINTS:

f = open(‘2tmonth.grib’)

grib_new_from_file

grib_get

max

grib_release

grib_new_from_file


f.close()
```

Let’s play with indexed access…

- Let’s try some **indexed** access to the grib file ztuv.grib:

1. Create the index for the file on the dataDate and shortName
2. Get the different dates and parameters available
3. Select one of the dates and one of the parameters
4. Get the handler for the matching message
5. Get the value of the ‘dataDate’ and ‘shortName’ parameter
6. Release the handler
7. Now select a different date and parameter and repeat from step 4
   - Do it for a couple of times at least
8. Release the index

```
HINTS:

grib_index_new_from_file

grib_index_get

grib_index_select

grib_new_from_index

grib_get

grib_release

grib_index_select


grib_index_release
```

Let’s modify a grib message

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

1. Open the input file
2. Get the handler for the first message in the file
3. Get the value of the ‘dataDate’ and ‘step’ parameter
4. Set the dataDate to 20120221 and step to 12
5. Open the output file for writing
6. Write the current modified message to the file
7. Release the handler
8. Close the output file
9. Close the input file
10. Check the new file with grib_ls

```
HINTS:

fin = open(‘2t.grib’)

grib_new_from_file

grib_get

grib_set

fout = open(‘2tmod.grib’,’w’)

grib_write

grib_release

fout.close()

fin.close()
```

Example scripts

- What is in the directories...
  - ecCodes:
    - 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
    - 2t.py, sst.py
  - magics: example of plotting using Magics++
    - basic_gribapi.py, basic_magics.py colour_gribapi.py magics.py
  - performance: little example comparing the performance of the tool, the Fortran and the python APIs

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

SciPy

http://www.scipy.org/

Matplotlib

http://matplotlib.sourceforge.net/

ecCodes

https://software.ecmwf.int/wiki/display/ECC/ecCodes+Home

Questions?