IPSL BootCamp: python

Institute Pierre Simone Laplace, IPSL BootCamp

The content of the BootCamp can be found in:
https://forge.ipsl.jussieu.fr/igcmg_doc/wiki/Train

March 21, 2016

Contents

1 Introduction **2**
1.1 Object 2

2 basics **3**
2.1 before starting 3
2.2 print 3
2.3 if 3
2.4 loops 4
2.5 list 4
2.6 dictionary 5
2.7 tuple 5
2.8 Miscellaneous 5
2.9 libraries 5

3 NO such basics **6**
3.1 functions 6
3.2 classes 7

4 netCDF & python **8**

5 drawing: matplotlib **9**

6 Useful links **10**

1 Introduction

Python was born on 1989 by 'Guido van Rossum' and takes its name from the *'Monthy Python Flying Circus'*[1]

It is a high level interpreted language, designed to be easy to be read and it is object-oriented.

There is a 'new' version of python (python 3.x) which is not compatible with the python.2.x. This course uses python 2.x. In a mid term is recommendable to migrate to python 3.x, nowadays (2016) might not be fully recommendable.

- **High level language:** Programmer does not need to worry to much about to deal with the CPU and memory. It is done by the program which it has a high level of abstraction. That facilitates the coding and reading the source, but penalize efficiency and speed
- **Interpreted language:** Code is not compiled. When on executes the source, it is read and executed
- **Objects:** Encapsulated data and methods in a single piece of code. Very versatile and flexible
- **easy to be read:** It is mandatory to indent the code

Python is of free access and it is managed by the *Python Software Foundation*.

1.1 Object

In the old class programming languages like: C, Fortran, bash, ... variables might be algebraic data-sets (scalar, vector, matrices) of 4 basic types: integer, float, boolean and character.

An object could be understood (in a very simple way) as a variable which has a more complex structure. It can be a combination of variables, functions and data-sets. Thus it is not only a group of values, either a memory address with a 'dynamic' behavior.

[1] https://en.wikipedia.org/wiki/Monty_Python

2 basics

2.1 before starting

Some very basic zero stuff:

- `#`: comment character
- `>>>`: Terminal string in an 'interactive' python session (after `$ python`)
- Mandatory to indent on `if`, `loop` (recommended 4 white spaces)
- `\`: Character of continuation of line
- Case Sensitive program
- Language based on the uploading of different packages via `import`
- Counts as C-like (starting at 0!)
- Extension of the scripts `.py`
- `[var][i]`: from variable `[var]`, access position `i`
- `[obj].[mth]`: from object `[obj]`, use the method `[mth]`

2.2 print

Some initial prints:

```
>>> print 'Hola'
```

prints string between ' '

```
Hola
```

Range print:

```
>>> print range(4)
```

Range of 4 values from 0 to 3

```
[0, 1, 2, 3]
```

2.3 if

'conditionals', a 'nice' if

```
i = 3
if i == 2:
   print 'two'
elif i == 3:
   print 'three'
else:
   print 'whatever'
```

':' to mark end of 'if'
'indent' mandatory !!
no-indent as it is part of the initial 'if'

No end if instruction, just do not indent

```
three
```

2.4 loops

Again, indented 'nice' loops

```
for i in range(4):
   print i
```

':' as end of loop instruction
indent to mark the loop

```
0
1
2
3
```

```
for i in range(1,10,2):
   print i
```

from '1' to '10' every '2'

```
1
3
5
7
9
```

2.5 list

'List' is a data structure of values sorted as they fill it. It can grow, and it can contain different types of data

```
>>> listvalues = [ ]
>>> listvalues.append('1')
>>> listvalues.append(2.)
>>> listvalues.append(3)
>>> listvalues.append(2)
>>> print listvalues
```

'[ ]' as list creation
giving the first value as string
second value as float
third value as integer

print values

```
['1', 2.0, 3, 2]
```

```
>>> print listvalues[2]/listvalues[1],
listvalues[2]/listvalues[3]
1.5 1
>>> print type(listvalues)
<type 'list'>
>>> print type(listvalues[0]),
type(listvalues[1])
<type 'str'> <type 'float'>
```

Tacking different values.

Result depends on type of variable
'type' to know the type of the list

'type' to know the type of the values within the list

Some methods of a list

```
>>> print listvalues.index(3)
2
>>> list2 = listvalues.pop(2)
>>> print list2
3
>>> print listvalues
['1', 2.0, 2]
```

print a single value of the list

removing the third value of the list
provides the removed value
final list without the third value

2.6 dictionary

Data-structure composed with pairs of values like in a dictionary with its 'word' and its 'meaning'. Values are not alphabetically sorted. It can contain different kind of data

```
>>> lmdcouloir = { }
>>> lmdcouloir['218'] = ['Shan', 'Lluis']
>>> lmdcouloir['313'] = 'seminar room'
>>> print lmdcouloir
{'313': 'seminar room', '218': ['Shan',
'Lluis']}
>>> print lmdcouloir['313']
seminar room
```

'{ }' definition of a variable as a dictionary
providing values for key '218' as a list
providing values for key '313' as a string
printing content of the dictionary

print content for the '313' value

Some methods of a dictionary:

```
>>> lmdcouloir.keys()
['313', '218']
>>> lmdcouloir.has_key('315')
False
```

listing all keys of the dictionary

search for the existence of a 'key'

2.7 tuple

Data-structure with values. It can not grow. Data access is more faster. It can mix data types

```
>>> tuplevalues = ()
>>> tuplevalues = (1, 2, 3, 89)
>>> print tuplevalues
(1, 2, 3, 89)
```

'( )' definition of variable as tuple
assigning values to tuple
printing values

2.8 Miscellaneous

Some general hints

- `>>> print dir([obj])`: list of the methods of `[obj]`
- `>>> print [obj].__doc__`: provides documentation of object `[obj]`
- `>>> quit()`: exiting python
- `"""`: for documentation in a class/function
- `'Header'` beginning of scripts, where one can import a huge number of different libraries/modules/packages

```
import [library]
import [library] as [abbreviation]
from [library] import [section]
```

Import all the content of the library
library's functions called as [abbreviation].[function]
just using a section of the library

2.9 libraries

python can be run with complementary powerful libraries. They cover a wide range of functinoalities and capabilities. A very short and the most valuable ones is here given:

- `numpy`, `scipy`: the scientific numerical libraries
- `os`, `subProcess`: for system calls
- `matplotlib`: for plotting (similar to 'matplot')
- `datetime`: for date/time variables
- `optparse`: parsing for arguments options when calling script

3 NO such basics

There are two main ways to generate self content actions: functions, classes

3.1 functions

The general structure of a function is this:

```
def FunctionName([Arg1], [Arg2], ...):
   """ Description of the function
   """

   (...)

   return [result of the function]
```

'def' marks definition of function ':' end of definition
""" to start documentation of the function
""" to end documentation of the function

whatever is required

values to provide as result of the function

A given example for the Fibonacci's series https://en.wikipedia.org/wiki/Fibonacci_number

```
def fibonacci(Nfibo):
   """ Function to provide the Nfibo
     numbers of the Fibonacci numbers
     Nfibo= Quantitty of numbers
     more info:
     http://en.wikipedia.org/wiki/Fibonacci_number
   N_i = N_i-2 + N_i-1
   """
   import numpy as np
   errmsg= 'ERROR - error - ERROR - error'

   fname = 'fibonacci'

   if Nfibo == 'h':
      print fname + '_______'
      print fibonacci.__doc__
      quit()

   if Nfibo < 2:
      print errmsg
      print ' ' +fname+ ':  Increase' + \
        Nfbio!'
      print ' only', Nfibo, 'given !!'
      quit(-1)

   numbers = np.zeros((Nfibo), dtype=int)

   numbers[0] = 0
   numbers[1] = 1

   for i in range(Nfibo-2):
      numbers[i+2]=numbers[i]+numbers[i+1]

   return numbers
```

Function definition
Retrieval of 'Nbio' numbers of Fibonacci

Getting module 'numpy' as np

Message to appear in case of help
is required when Nfbio='h'
here will be printed the header content within """

Error message if 'Nfbio' is too small

Error message system output '-1'

numpy array of 'Nfbio' zeros

First value '0'
Second value '1'

Loop over the 'Nfbio-2' numbers

Returning the numpy array

When we call:

```
>>> fibonacci('h')
fibonacci_______
Function to provide the Nfibo
numbers of the Fibonacci numbers
  Nfibo= Quantitty of numbers
  more info:  h
  ttp://en.wikipedia.org/wiki/Fibonacci_number
  N_i = N_i-2 + N_i-1

>>> fibonacciSeries = fibonacci(7)
>>> print fibonacciSeries
[0 1 1 2 3 5 8]
```

Regarding function documentation

Function brings back a numpy array
it has to be printed to see its values!

3.2 classes

'classes', are similar to functions, but much more complex and flexible (too much for an introductory course!)

4 netCDF & python

There are different libraries to access netCFD. In this case we will use `http://netcdf4-python.googlecode.com/svn/trunk/docs/netCDF4-module.html`. Very flexible, robust and easy to use

- Some basic functions
  - `[ncobj] = NetCDFFile([filename], '[access]')`: open a file `[filename]` as object `[ncobj]` with `[access]`='`w`', write; '`a`', append, '`r`', read
  - `[ncobj].variables`: dictionary will all the variables of the file
  - `[ncobj].dimensions`: dictionary will all the dimensions of the file
  - `[ncvarobj] = [ncobj].variables['varname']`: object `[ncvarobj]` with variable 'varname'
  - `[ncvarobj].dimensions`: dictionary will all the dimensions of the variable 'varname'
  - `[ncvarobj][:]`: all the values of variable 'varname' (shape automatically given)
  - `[ncvarobj].shape`: shape of the variable 'varname'
  - `createVariable([varname], [kind], [dimensions], fill_value=[missValue])`: creation of a variable with a missing value of `[missValue]`
- Example with '`reading_nc.py`'

Code	Comment
`import numpy as np` `from netCDF4 import Dataset as NetCDFFile`	Importing 'netCDF4' module to manage netCDF
`filename = 'Relief.nc'`	name of the file
`toponame = 'RELIEF'`	name of the variable with topography
`ncobj = NetCDFFile(filename, 'r')`	open netCDF in read mode
`objtopo = ncobj.variables[toponame]`	getting topography variable object
`print 'variables:',ncobj.variables.keys()`	printing all variables within the file
`print 'dimension of the topography:', \` `  objtopo.shape`	
`topovals = objtopo[:]`	getting topography values
`utopo = objtopo.getncattr('units')`	getting the 'units' as an attribute
`print 'max.  height:', np.max(topovals), \` `  utopo`	computing the maximum height
`ncobj.close()`	

When we execute

```
$ python reading_nc.py
variables:  [u'longitude', u'latitude', u'RELIEF']
dimension of the topography:  (1080, 2160)
max.  height:  7833.0 m
```

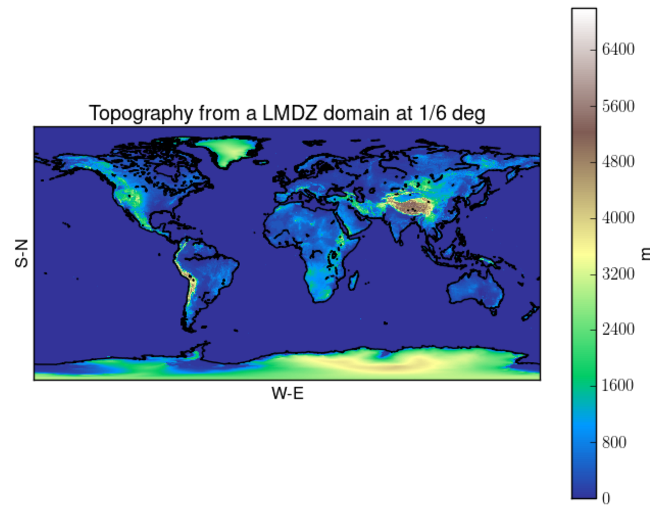
5 drawing: matplotlib

For plotting there is a matplot like environment http://matplotlib.org/. Very flexible, robust and powerful. Example with 'plotting_topo.py'

```
import numpy as np
from netCDF4 import Dataset as NetCDFFile
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

filename = 'Relief.nc'
toponame = 'RELIEF'
ncobj = NetCDFFile(filename, 'r')

objtopo = ncobj.variables['toponame']
olon = ncobj.variables['longitude']
olat = ncobj.variables['latitude']
topovals = objtopo[:]
utopo = objtopo.getncattr('units')

plt.rc('text', usetex=True)

dx = olon.shape[0]
dy = olat.shape[0]
nlon = np.min(olon)
xlon = np.max(olon)
nlat = np.min(olat)
xlat = np.max(olat)
plt.xlim(nlon, xlon)
plt.ylim(nlat, xlat)
m = Basemap(projection='cyl', \
  llcrnrlon=nlon, llcrnrlat=nlat, \
  urcrnrlon=xlon, urcrnrlat= xlat, \
  resolution='l')

lons = olon[:]
lats = olat[:]
x,y = m(lons,lats)

plt.pcolormesh(x,y,topovals, \
  cmap=plt.get_cmap('terrain'), \
  vmin=0., vmax=7000.)
cbar = plt.colorbar()
cbar.set_label(utopo)
m.drawcoastlines()

plt.xlabel('W-E')
plt.ylabel('S-N')
plt.title('Topography from a ' + \
  'LMDZ domain at 1/' + \
  str(olon.shape[0]/360) + ' deg')
plt.savefig("topo.png")
plt.close()
```

importing 'matplotlib'
importing coastal line for mapping
opening for reading netCDF
getting topography variable as an object
getting longitudes variable as an object
getting latitude variable as an object
getting topography values
using LaTeX like format for strings in the plot
x-range of the values
y-range of the values
minimum of the longitudes
maximum of the longitudes
minimum of the latitudes
maximum of the latitudes
x-limits of the plot
y-limits of the plot
using 'Basemap' to geo-localize the values
in a cylindric projection
with low ('l') resolution information
getting longitudes (in file is 1D)
getting latitudes (in file is 1D)
lon, lat 2D matrix plotting creation
plotting using shading of the topography
'x', 'y', coordinates, 'topovals' values
'terrain' color bar within the range [0, 7000.]
drawing color bar
giving units to the color bar
drawing coastal-line
x-axis label
y-axis label
title of the graph
output as Portable Network Graphics (png)
finishing graphic

Which generates:



6 Useful links

- Python Source: https://www.python.org/
- Shan's python course: http://www.xn--llusfb-5va.cat/python/ShanCourse/PythonCourse.html (almost the same content as here)
- List of some useful modules: https://wiki.python.org/moin/UsefulModules
- Aymeric Spiga's PLANETOPLOT plotting toolkit from LMD useful to plot http://www.lmd.jussieu.fr/~aslmd/planetoplot
- L. Fita's netCDF/plottiong toolkit from LMD for netCDF file managements is already available from `LMDZ_WRF` subversion trunk repository:

```
$ svn co http://svn.lmd.jussieu.fr/LMDZ_WRF/trunk/tools LFita_pyTools
```

  - `nc_var.py`: wrapper to use all functions:

```
$ python nc_var.py -o [operation] -f [file] -S [values]
```

  - `nc_var_tools.py`, all functions and classes which are used throughout `nc_var.py`
  - `drwaing.py`: wrapper to use all functions:

```
$ python drawing.py -o [graphic] -f [file] -S [values]
```

  - `drawing_tools.py`, all functions and classes which are used throughout `drawing.py`
  - `variables_values.dat`, necessary external ASCII file with CD-definition of variables