Python and Statistics for Matlab Users

Session 2

Nicolas Barrier

OT-Med Labex

September 19, 2016

Table of Contents

String manipulation

```
string1 = 'Char1 Char2 Char3'
chars = list(string1) # returns list of char ["S", "t" ... ]

# Replace "Char2" by "toto" on "string1"
string2 = string1.replace('Char2', 'toto') # 'Char1 toto Char3'

# splits the string into words given a separator
sep = ' '
words = string1.split(sep) # ['Char1', 'Char2', 'Char3']

# concatenates a list of string into one string, separated
# by a given separator
sep = '\t'  # tabulation
string3 = sep.join(['toto1', 'toto2', \
                    'toto3']) # 'toto1   toto2   toto3'
```

String manipulation

```
string4 = 2*'toto1' + '\t' + 'toto2' + \
          '\n' + 'toto3 ' + str(10)
# \n: linebreak
#"toto1toto1      toto2
# toto3 10"

# REMEMBER: strings are viewed as a list of characters

# Matlab's sprintf
x = 10; y = 0.5; z = 0.005
string5 = '%04d, %.3f, %.5f\n' %(x, y, z)
# 0010, 0.500, 0.00500
#

# remove the \n character (when reading a file)
string6 = string5.strip()
# 0010, 0.500, 0.00500
```

Table of Contents

Functions

```
def function(arg1, arg2, arg3=0, arg4=-1):
    # arg1, arg2: compulsory arguments
    # arg3, arg4: optional arguments (def. are 0 and -1, resp.)
    output = arg1 + arg2 + arg3*(arg1+arg2) \
        + arg4*(arg1-arg2)
    return output, output**2

x1, x2 = function(1, 2) # x1=4, x2=16 (floats)
x1, x2 = function(1, 2, arg4=1) # x1=2, x2=4
x1, x2 = function(1, 2, arg3=1, arg4=1) # x1=5, x2=25
x = function(1, 2, arg3=1, arg4=1) # x=(5, 25): tuple
x[0], x[1] # 5, 25
```

Note

A `tuple` is a `list` that you cannot modify.

Functions: the *args argument

When the number of arguments is variable, you can use the *args argument, which is a list of arguments.

```
def function2(x, *args):

    # if additional args. are provided
    # return a tuple
    if len(args) > 0:
        return x, args
    # else return a single variable
    else:
        return x

function2(3) # 3
function2(3, 'toto', 5.4) # (3, ['toto', 5.4])
```

Functions: the **kwargs argument

Imagine you want that your function takes as many arguments as the `pylab.plot` function. This is achieved by using the `**kwargs` argument, which is a dictionnary of arguments:

```
def function3(x, **kwargs):

    y = np.cos(x)
    plt.figure()
    plt.plot(x, y, **kwargs)

function3(x) # no additional arguments
# kwargs provided as key,val couples
function3(x, color='r', linewidth=1)
# kwargs provided as a dict
argsdict = {'linewidth':4, 'color':'orange', 'linestyle':'--'}
function3(x, **argsdict)
```

Functions: the **kwargs argument

It is possible to define several **kwargs by using dict.

```
def function4(x, dictfig={}, dictplt={}):
    y = np.cos(x)
    plt.figure()
    plt.plot(x, y, **dictplt)
    plt.savefig("figure.png", **dictfig)

# extra arguments for the plot function
argsplt = {'linewidth':4, 'color':'orange', 'linestyle':'--'}
# extra arguments for the savefig function
argsfig = {'facecolor':'gray'}

function4(x, dictfig=argsfig, dictplt=argsplt)
```

Table of Contents

Reading a file

```
from scipy.io.matlab import mio

# open the file as a dictionnary
fin = mio.loadmat('Keeling_MaunaLoa_CO2_interp.mat', \
                  squeeze_me=False)
# squeeze_me=optional argument (if False, keeps Matlab's format)

fin.keys()
# ['time','__version__','__header__','CO2','__globals__']

time = fin['time'] # time.shape=(1, 456): Matlab's format
co2 = fin['CO2'] # co2.shape=(1, 456)

time = np.squeeze(time) # time.shape=(456)
co2 = np.squeeze(co2) # co2.shape=(456)
```

Writting a file

```
# to save data in a matfile:
x = np.linspace(0, 2*np.pi, 200)
y = np.cos(x)

# convert data into a dictionnary
output = {'datax':x, 'datay':y}

# saves the dictionnary into a file
mio.savemat('myfile.mat', output)
```

Warning!

Does not work with `masked` arrays. Tip: first convert masked values into `np.nan`. Then convert the `numpy.ma` arrays into `numpy` arrays.

Table of Contents

Example

The NAO index:

Hurrell PC-Based North Atlantic Oscillation Index (Monthly)

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1899	-0.06	-0.70	-1.05	-0.42	-0.60	0.15	0.22	-0.59	0.50	0.20	0.60	-1.07
1900	0.34	-2.53	-1.34	0.53	0.03	-0.25	-0.39	-0.19	0.72	0.16	0.11	0.78
1901	0.26	-2.06	-0.42	0.30	-0.49	0.41	-0.05	0.14	-0.07	0.34	-0.93	-0.41
1902	0.26	-2.37	-0.16	-0.83	0.06	-1.32	-0.59	-0.64	-0.55	0.22	-0.23	0.39
1903	0.23	2.22	2.22	-0.53	0.25	-1.31	-0.49	-0.08	-0.28	-0.57	0.63	-0.19
1904	0.88	0.10	0.12	1.63	0.49	-0.10	-0.21	-0.34	-0.06	0.44	-0.27	-0.17
1905	0.87	1.16	1.17	-0.88	0.62	-0.05	0.07	-0.12	0.11	-0.78	-0.01	0.92
1906	1.03	0.69	0.33	0.86	-0.95	-0.56	0.35	-0.62	0.03	0.56	-0.96	0.40

and so on until 2016. . .

Reading the text file

```
# open the file in read mode
fin = open('../../data/nao_pc_monthly_0.txt','r')

# read all the lines
lines = fin.readlines()
# closing the file
fin.close()

type(lines) #<type 'list'> list object
len(lines) #116
lines[0] # "  Hurrell PC-Based [...]"
lines[1] # " Jan    Feb    Mar [...]"
type(lines[0]) # <type 'str'> string types
```

Extracting the data

```
data = [] # initialize data list
# loop over the lines, starting on line 2 to avoid the header
for l in lines[2:]:
    sp = l.split() # split the string into an array of strings
    # ['1899', '-0.05' ... ]
    data.append(sp) # add this list to the data list

# convert the list into an array
data = np.array(data) # (115, 13)
type(data) # <type 'numpy.ndarray'>
data.dtype # |S5 : string array

year = data[:, 0].astype(np.int) # year is the first column
nao = data[:, 1:].astype(np.float) # nao index -> other columns
```

Regular expressions

Regular Expressions allow to find the lines that match a certain pattern. Imagine you have a corrupted NAO file:

```
Hurrell PC-Based North Atlantic Oscillation Index (Monthly)
        Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct    Nov    Dec
        1899  -0.06  -0.70  -1.05  -0.42  -0.60   0.15   0.22  -0.59   0.50   0.20   0.60  -1.07
        1900   0.34  -2.53  -1.34   0.53   0.03  -0.25  -0.39  -0.19   0.72   0.16   0.11   0.78
        1901oups 0.26  -2.06  -0.42   0.30  -0.49   0.41  -0.05   0.14  -0.07   0.34  -0.93  -0.41
        1902   0.26  -2.37oups  -0.16  -0.83   0.06  -1.32  -0.59  -0.64  -0.55   0.22  -0.23   0.39
        1903   0.23   2.22   2.22  -0.53   0.25  -1.31  -0.49  -0.08  -0.28  -0.57   0.63  -0.19
        1904   0.88   0.10   0.12   1.63   0.49  -0.10  -0.21  -0.34  -0.06   0.44  -0.27  -0.17
```

Regular expressions

```
import re  # package for regular expressions

# pattern to match
pattern = '^[0-9]{4}( +-?[0-9]+.[0-9]{2}){12} *$'
#^: line beginning. [0-9]{4}: 4 integers. $: line end
# +: one or more of spaces. [0-9]+: 1 or more int
# (pattern){12}: 12 repeats of the pattern in brackets

regexp = re.compile(pattern)  # compilation of the pattern

data = []
for l in lines:
    if regexp.match(l): # if the pattern is matched
        data.append(l.split()) # adding the data into the list
    else:
        print "line discarded: %s" %l
```

Writting a file: Example

We will write out the results of the `cos`, `sin` and `tan` functions for 5 values of `x` between 0 and $\pi/4$.

```
x       cos     sin     tan
0.00000000      1.00000000      0.00000000      0.00000000
0.19634954      0.98078528      0.19509032      0.19891237
0.39269908      0.92387953      0.38268343      0.41421356
0.58904862      0.83146961      0.55557023      0.66817864
0.78539816      0.70710678      0.70710678      1.00000000
```

```
# generating the data
xdata = np.linspace(0, np.pi/4., 5)
cosx = np.cos(xdata)
sinx = np.sin(xdata)
tanx = np.tan(xdata)
```

Writting the file

```
# opening the file
fout = open('outfile.txt', 'w')

# writting a header: 4 strings sep. by tabs.
string='%s\t%s\t%s\t%s\n' % ('x', 'cos', 'sin', 'tan')
fout.write(string)

# writting the data: four floats sep. by tabs.
for x, c, s, t in zip(xdata, cosx, sinx, tanx):
    string = '%.8f\t%.8f\t%.8f\t%.8f\n' %(x, c, s, t)
    fout.write(string)

# closing the file
fout.close()
```

Table of Contents

Reading a file: Example

```
netcdf regimes_natl {
dimensions:
        cluster = 4 ;
        year = UNLIMITED ; // (53 currently)   record dimension
variables:
        int cluster(cluster) ; variable type and name
        int occu_yr(year, cluster) ;
                occu_yr:long_name = "Number of occurrence per year" ;
                occu_yr:_FillValue = -2147483647 ; variable attributes
        int year(year) ;
                year:long_name = "YYYY year" ;

// global attributes:
                :creation_date = "Fri Oct 11 01:14:13 CEST 2013" ;  file attribute
                :title = "Regimes statistics" ;
                :script_name = "/home/badoit/barrier/NCL/
                Classification/compute_regime_stat.ncl" ;
}
```

Reading a file

```
# import the relevant package
from scipy.io.netcdf import netcdf_file

# open the file. Put mmap=True if you only want to access
# data directly from file (no writting of the variable)
fin = netcdf_file('regimes_natl.nc', 'r', mmap=False)

# retrive the global attributes
creation_date = fin.creation_date
script_name = fin.script_name
```

Reading a file

```
# retrive the variable
occu_yr = fin.variables['occu_yr']
type(occu_yr) # <type 'netcdf_file.Variable'>

# retrive the variable attribute
long_name = occu_yr.long_name

# retrive the values
occu_yr = occu_yr[:]
type(occu_yr) # <type 'numpy.ndarray'>
occu_yr.shape # (53, 4)
# Here, no more access to the attributes since it
# is a numpy array!
```

Reading multiple files

```
from glob import glob # import of the glob module

finlist = np.sort(glob('../../data/ISAS*nc'))
# sorted list of input files

# loop over all the files
for finname in finlist:
    fin = Dataset(finname, 'r') # opening of the files

    if finname == finlist[0]:  # if first file opened
        output = fin.variables['TEMP'][:] # init of output array

    else:
        temp = fin.variables['TEMP'][:]
        # cont. of array
        output = np.concatenate((output, temp), axis=0)

    fin.close()
```

Reading multiple files

Hint!

Opening multiple files is easy with the `MFDatset` function of the `netCDF4` library:

```
from netCDF4 import MFDataset
fin = MFDataset("../../data/ISAS*")  # no 'r' arg

output = fin.variables['TEMP'][:]

time = fin.variables['time']
tunits = time.units
time = time[:]

fin.close()
```

Writting a file: Example

```
netcdf test_netcdf {
dimensions:
    time = UNLIMITED ; // (1 currently)
    x = 20 ;
    y = 30 ;
variables:
    float y(y) ;
    float x(x) ;
    float data(time, x, y) ;
        data :description = "Array of ones" ;
    int time(time) ;
        time:units = "days since 1950-01-01T00:00:00Z";

// global attributes:
    :date = "14/07/2016" ;
    :scriptfile = "/Users/Nicolas/Dropbox/python_class/day2/
                  programs_session2/write_netcdf.py" ;
```

Writting a file

```
import numpy as np
from scipy.io.netcdf import netcdf_file
import time  # time library
import os.path

# generating data
nt = 1; nx = 20; ny = 30
t = np.arange(nt)
x = np.arange(nx)
y = np.arange(ny)
data = np.ones((nt, nx, ny))

foutname = 'test_netcdf.nc' # file name
fout = netcdf_file(foutname, 'w') # open the file in write mode
```

Writting a file

```
# writting global attrs.
# script name + current time
fout.scriptfile = os.path.abspath(__file__) # script path
fout.date = time.strftime("%d/%m/%Y") # current date

# creating dims
fout.createDimension('time', None) # record dim: dim=None
fout.createDimension('x', nx)
fout.createDimension('y', ny)

# creating the variable
fout.createVariable('time', 'i', ('time',)) # time as int
fout.createVariable('x', 'f', ('x',)) # x as float
fout.createVariable('y', 'f', ('y',))
fout.createVariable('data', 'f', ('time', 'x', 'y'))
```

Writting a file

```
# writting the data
var = fout.variables['data'] # netcdf object
var.description = "Array of ones" # write attr.
var[:] = data # write data. dont forget the [:] !

var = fout.variables['time']
var.units = 'days since 1950-01-01T00:00:00Z' # attr
var[:] = t  # data

fout.variables['y'][:] = y # write data
fout.variables['x'][:] = x # write data

fout.close() # closing the file
```

Table of Contents

Time handling

```
from datetime import datetime
import time

# print current time as a string
time.strftime('%d %b %Y %H:%M:%S')
# 14 Jul 2016 11:36:16

# initialize a datetime object
date = datetime(2014, 1, 4, 7, 03, 00) # 2014-01-04 07:03:00
date.year # 2014
date.month # 1
date.day # 4 etc...
date.strftime('%d %b %Y %H:%M:%S') # 04 Jan 2014 07:03:00
```

NetCDF time

```
from netcdftime import utime
from scipy.io import netcdf_file

# opening the file, extracting the time units and the time val.
fin = netcdf_file('../../data/ISAS13_20120115_fld_TEMP.nc','r')
time = fin.variables['time'] # netcdf object
units = time.units
time = time[:] # array!

# object for time/date conversions
cdftime = utime(units, calendar='gregorian')

date = cdftime.num2date(time) # conversion from num. to date
# [datetime.datetime(2012, 1, 15, 0, 0)]: array!

# conversion from date to num.
datebis = [datetime(2012,1,30,0,0),datetime(2013,1,30,0,0)]
timebis = cdftime.date2num(datebis) # [22674.0 23040.]
```