Python Crash Course for Climate Scientists

Oliver Angélil (molofishy@gmail.com) and Nadja Herger (nadja.herger@student.unsw.edu.au)

May 2016

1 Setup

1.1 Download and Install Python

Download the miniconda distribution of Python 2.7 from here.

Navigate to the file in bash, and type:

```
$ bash Miniconda-latest-MacOSX-x86_64.sh
```

(If you're not on a Mac, replace `Miniconda-latest-MacOSX-x86_64.sh` with the relevant file name you just downloaded.)

Follow the on-screen instructions. Once you're done, a 'miniconda' directory should appear in your home directory, and a line in your `.bash_profile` should declare 'miniconda' as a path.

1.2 Download Packages

Miniconda comes with a package manager called 'conda'. There are thousands of python packages available, but the most useful packages for a climate scientist are:

- netcdf4 (read & write NetCDF files)
- numpy (manipulation of arrays and maths functions to operate on them)
- matplotlib (plotting)
- basemap (plotting data on map projections)

Download them with:

```
$ conda install netcdf4 numpy matplotlib basemap
```

List the installed packages in your conda environment:

```
$ conda list
```

2 Reading in a NetCDF file

First download an example NetCDF file from Oliver's website:

```
$ wget http://web.maths.unsw.edu.au/~oangelil/downloads/precip.nc -P ~/
# In case wget doesn't exist on your laptop, you can install it with: sudo port install wget
# once you have installed MacPorts.
# Alternatively, pasting the link above into your browser should automatically start the
# download of the NetCDF file
```

Create a new file called 'first_script.py' and add these lines to it:

```
# Load modules
from netCDF4 import Dataset # to work with NetCDF files
import numpy as np
import matplotlib.pyplot as plt # to generate plots
from mpl_toolkits.basemap import Basemap # plot on map projections
from os.path import expanduser
home = expanduser("~") # Get users home directory
import os # operating system interface

# Input NetCDF file info
file = home+'/precip.nc'
variable = 'precip'

# Extract the variables
fh = Dataset(file, mode='r') # file handle, open in read only mode
lon = fh.variables['lon'][:]
lat = fh.variables['lat'][:]
pr = fh.variables[variable][:]
fh.close() # close the file
```

Execute the file in a new terminal tab as an interactive session with:

```
  $ python -i first_script.py #to exit interactive mode, type ctrl+d.
```

While in the interactive session, type the following to see the dimensions of the variables:

```
pr.shape # Results in nmonths x nlat x nlon (811, 72, 144)
len(lat) # Length of lat is 72
print 'The variable contains ' + str(pr.shape[0]) + ' timesteps.' # Print statement to the screen
```

3 Accessing the data

```
# Outputting all the latitude values
lat # prints a list to the screen with values from 88.75 to -88.75

# Accessing the first and the last value of lat
lat[0] # Caution! Python's indexing starts with zero
lat[-1] # gives the last value of the vector

# Getting the precipitation field of the first month
pr[0,:,:] # or simply pr[0]
pr[0].shape # (72, 144)
```

4 Getting help

If you want to learn more about what a specific function does, you can use the help function:

```
help(np.mean)
```

Exit the help page by pressing "q". You can also apply the help function on an object (variable, list, ...) to learn more about what you could do with it.

```
a = 77
help(a)
b = ['a',334,'abc']
help(b)
```

You can use the “dir” function on a function if you are interested in the “sub-functions” available in the master-function.

```
dir(np)
```

You can also try to find solutions to any problem in the python documentation: https://www.python.org/doc/versions/

5 Visualising the Data

5.1 Quick and Dirty Visualisation

```
plt.imshow(pr[0]) #data for the first time-step (January 1948)
plt.show()
```

5.2 Visualising on a Map Projection

```
lons, lats = np.meshgrid(lon,lat)
m = Basemap(projection='kav7',lon_0=0)
m.drawmapboundary(fill_color='Gray', zorder=0)
m.drawparallels(np.arange(-90.,99.,30.), labels=[1,0,0,0])
m.drawmeridians(np.arange(-180.,180.,60.), labels=[0,0,0,1])
h=m.pcolormesh(lons,lats,pr[0], shading='flat',latlon=True)
m.colorbar(h,location='bottom',pad="15%", label='Rainfall [mm/day]')
plt.ion(); plt.show()

#Uncomment this line to save the figure to your home directory:
#plt.savefig(home+'/basemap_figure.png')
```

6 Working with masked arrays

In the previous section we noticed that our precipitation data-set only has values over land. This is where masked arrays come into play.

```
# Have a look at pr
pr # Python automatically created a masked array with the data and the mask

# Have a look at the data and the mask separately
pr.data # gives you the data array only
pr.mask # gives you True where the data is masked (over the ocean) and False where data is available (over land)

# Create a vector of precipitation data where values are not masked
pr.data[~pr.mask] # the tilde inverts the mask

# Create a climatology field from the precipitation data
clim = np.mean(pr, axis=0) # because pr is a masked array, all masked values are ignored in the calculation.
#‘clim’ can be plotted onto a map projection the same way you plotted ‘pr[0]’ above.
```

7 Calculate and plot the global monthly mean precipitation rate

```
# Define weighting matrix
wgtmat = np.cos(np.tile(abs(lat[:,None])*np.pi/180, (1,len(lon)))) # Dimension: 72x144
```

```
# Calculate the global monthly mean precipitation rate
mean_pr = np.zeros(pr.shape[0]) # Preallocation
for i in range(pr.shape[0]): # Don't forget the ':'
    mean_pr[i] = np.sum(pr[i] * wgtmat * ~pr.mask[i])/np.sum(wgtmat * ~pr.mask[i])

# Closer look at the range function in the loop
range(pr.shape[0]) # vector from 0 to 810

# Example of a simple if-statement
if variable == 'precip':
    unit = 'mm/day'
elif variable == 'tas':
    unit = 'degC'
else:
    print 'Unknown variable.'

# Plot a time series of monthly rainfall
fig=plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
plt.plot(range(1,pr.shape[0]+1), mean_pr, 'k-', label='precip',linewidth=2)
plt.xlabel('Month')
plt.ylabel('Precipitation Rate [' + unit + ']')
plt.axis([1,pr.shape[0]+1, np.min(mean_pr)-0.05, np.max(mean_pr)+0.05])
plt.title('Monthly Mean Precipitation Rate Over Land',fontsize=12)
plt.legend(loc=2, borderaxespad=1., frameon=False)
plt.ion(); plt.show()
plt.savefig(home+'/monthly_global_precip.png')

# Calculate annual mean precipitation rates
mean_pr_reshaped = mean_pr[0:804].reshape((67,12))
mean_pr_yr = np.mean(mean_pr_reshaped,axis=1)
mean_pr_yr # print the results to the screen
```

8 Saving numpy arrays into a single file

```
# Saving
np.savez(home + '/outfile.npz', lat=lat, mean_pr_yr=mean_pr_yr)

# Loading
npzfile = np.load(home + '/outfile.npz')
npzfile.files # results in ['lat', 'mean_pr_yr']
npzfile['lat']
```

9 List comprehension

Often, list comprehensions can be used instead of for-loops. A simple example is shown below:

```
words = ['one','two','three','four']
uppercase_words = [words[i].upper() for i in range(len(words))]
#the same can be done with the following:
uppercase_words = [i.upper() for i in words]
```

10 Using CDO in Python (the CDO module)

There is also the possibility to work with the CDO module within Python. Many Python distributions (including anaconda) come with a command line tool called pip. Install the CDO module by typing the following in bash:

```
pip install --user cdo
```

Import the module into Python with:

```
# Load required module
from cdo import *
cdo = Cdo()

# Compute the global mean monthly precipitation (as in section 7) and return it as a numpy array:
mean_pr = np.squeeze(cdo.fldmean(input=file, returnArray='precip'))

#Use more than one CDO command and compute global precipitation anomalies
#relative to the 1971-2000 mean global precipitation:
global_pr_anomalies = np.squeeze(cdo.sub(input = '-fldmean ' + file + '
-timmean -selyear,1971/2000 -fldmean
' + file, returnArray='precip', options = '-L'))
#Sometimes CDO throws an error ("segmentation error"). Using the "-L" option can prevent this.

#Read in all years in the netCDF file:
years = cdo.showyear(input=file)
#this is one long string, so split each year into individual elements:
years = np.array(years[0].split(), dtype=np.float)
```

Detailed information on using this module can be found here. A useful reference card with important CDO commands can be found here. See some specific questions I asked in the CDO forums about using this Python module. For example how to use operators that are followed by a comma and a value, like the “runmean”, “remapcon”, and “selyear” operators, and how to use any of the operators beginning with “show”.