Training Objectives

We want to introduce:

- **Basic concepts of Python programming**
- **Array manipulations**
- Handling of files
- 2D visualization
- EOFs
Based on the feedback we have received so far, we plan to have a hand-on presentation on the following topic(s):

**F2Py:** Python interface to Fortran
Tentative Date: April 29, 2013 at 1:30pm
Obtaining the Material

Slides for this session of the training are available from:

https://modelingguru.nasa.gov/docs/DOC-2322

You can obtain materials presented here on discover at

/discover/nobackup/jkouatch/pythontraininggsfc.tar.gz

After you untar the above file, you will obtain the directory *pythonTrainingGSFC/* that contains:

Examples/
Slides/
We installed a Python distribution. To use it, you need to load the modules:

```
module load other/comp/gcc-4.5-sp1
module load lib/mkl-10.1.2.024
module load other/SIVO-PyD/spd_1.7.0_gcc-4.5-sp1
```
What Have We Learned So Far?

Strings
'spam', "guido's"

Lists
[1, [2,'tree'], 4]

Dictionaries
'food': 'spam', 'taste':'yum'

Tuples
(1,'spam', 4, 'U')

NumPy Arrays
arange(a, b, m)
linspace(a, b, n)
array(list)
What Will be Covered Today

1. Matplotlib
   - 2D Plot
   - 3D Plot
   - Basemap toolkit

2. netCDF4

3. H5Py

4. Visualization Session
Matplotlib
Useful Links for Matplotlib

- **Video Presentation**
  
  http://videolectures.net/mloss08_hunter_mat

- **User’s Guide**
  
  http://mural.uv.es/parmur/matplotlib.pdf

- **Image Gallery**
  
  http://matplotlib.sourceforge.net/gallery.html
What is Matplotlib?

- Library for making 2D plots of arrays in Python
- Makes heavy use of Numpy and other extension code to provide good performance
- Can be used to create plots with few commands
What Can we Do with Matplotlib?

You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code.
Two Main Interfaces of Matplotlib

**pyplot**

- Provides a Matlab-style state-machine interface to the underlying object-oriented plotting library in matplotlib.
- Preferred method of access for interactive plotting.

**pylab**

- Combines the pyplot functionality (for plotting) with the Numpy functionality (for mathematics and for working with arrays) in a single namespace, making that namespace (or environment) even more Matlab-like.
- Formerly preferred method of access for interactive plotting, but still available.
pyplot vs. pylab

**pyplot:**

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

x = np.arange(0, 10, 0.2)
y = np.sin(x)

pyplot.plot(x, y)
pyplot.show()
```

**pylab:**

```python
from pylab import *

x = arange(0, 10, 0.2)
y = sin(x)

plot(x, y)
show()
```
# Syntax for Plotting

```python
#!/usr/bin/env python
import matplotlib.pyplot as plt

x = [...] # define the points on the x-axis
y = [...] # define the points on the y-axis

plt.plot(x,y) # display the plot on the screen
plt.show() # display the plot on the screen
```
Creating a Basic Graph

```python
#!/usr/bin/env python
import matplotlib.pyplot as plt

x = [2, 3, 5, 7, 11]
y = [4, 9, 5, 9, 1]
plt.plot(x, y)
plt.show()
```
Basic Graph
Some pyplot Functions

plot(x,y)                                # label the x-axis
xlabel('string')                        # label the y-axis
ylabel('string')                        # write the title of the plot
title('string')                         # adds grid boxes
grid(true/false)                        # type can be png, ps, pdf, etc
savefig('fileName.type')               # display the graph on the screen
show()                                  # set/get the xlimits
xlim(xmin,xmax)                         # set/get the ylimits
ylim(ymin,ymax)                         # to overlay figures on the same graph
hold(True/False)
Code for Plotting the Cosine Function

```python
#!/usr/bin/env python
import math
import numpy as np
import matplotlib.pyplot as plt

# Code for plotting the cosine function

t = np.arange(0.0, 1.0+0.01, 0.01)
s = np.cos(2*2*math.pi*t)
plt.plot(t, s)
plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('About as simple as it gets, folks')
plt.grid(True)
plt.savefig('simple_plot')
```

J. Kouatchou and H. Oloso (SSSO)
Simple Cosine Plot

About as simple as it gets, folks

voltage (mV)

time (s)
Two Figures on the Same Plot

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

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```
Graph of Two Figures on the Same Plot

Matplotlib

2D Plot

Graph of Two Figures on the Same Plot
figure(num)
    # allows to plot multiple figures at the same time
    # can be called several times
    # num: reference number to keep track of the figure object

subplot(numrows, numcols, fignum)
    # fignum range from numrows*numcols
    # subplot(211) is identical to subplot(2,1,1)
Sample Code for Plotting Four Figures and Axes

```python
plt.subplot(2,2,1)
plt.plot(x,y01,linewidth=3); plt.hold(True)
plt.plot(x,y02,'r',linewidth=3)

plt.subplot(2,2,2)
plt.plot(y03,linewidth=2)

plt.subplot(2,2,3)
plt.plot(x,y04,'k',linewidth=3); plt.hold(True)
plt.plot(x,y05,'--',linewidth=3)
plt.plot(x,y06,'r',linewidth=2)

plt.subplot(2,2,4)
plt.plot(Y04,linewidth=2.5)
```
Example of Graph with Four Figures on the Same Plot
Sample Pie Chart

```python
figure(1, figsize=(6,6))
ax = axes([0.1, 0.1, 0.8, 0.8])

labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
fracs = [15, 30, 45, 10]

explode=(0, 0.05, 0, 0)

pie(fracs, explode=explode, labels=labels)
```
Graph for a Pie Chart

Raining Hogs and Dogs

- Hogs: 30.0%
- Frogs: 15.0%
- Logs: 10.0%
- Dogs: 45.0%
import numpy as np
import matplotlib.pyplot as plt

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, 
    facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100, \ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
Graph for an Histogram

Histogram of IQ

\[ \mu = 100, \sigma = 15 \]
Using Mathematical Expressions in Text

- Matplotlib accepts TeX equation expressions in any text.
- Matplotlib has a built-in TeX parser
- To write the expression $\sigma_i = 15$ in the title, you can write:
  ```python
  plt.title(r'$\sigma_i=15$')
  ```
  where `r` signifies that the string is a raw string and not to treat backslashes and python escapes.
Sample Code for Annotating Text

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

ax = plt.subplot(111)
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black',
                             shrink=0.05), )

plt.ylim(-2,2)
plt.show()
```
Graph for Annotating Text
Log Plots

Use the following pyplot functions:

```python
semilogx() # make a plot with log scaling on the x axis
semilogy() # make a plot with log scaling on the y axis
loglog()   # make a plot with log scaling on the x and y axis
```
Graph with Log Plots

- **Semilogy**
  - Y-axis: Log scale
  - X-axis: Linear scale

- **Semilogx**
  - Y-axis: Linear scale
  - X-axis: Log scale

- **Loglog base 4 on x**
  - Both axes: Log scale

- **Errorbars go negative**
  - Y-axis: Log scale
  - X-axis: Log scale
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0.0, 1.01, 0.01)
s = np.sin(2*2*np.pi*t)

plt.fill(t, s*np.exp(-5*t), 'r')
plt.grid(True)
plt.show()
Graph for Plot with Fill
Legend

Call signature:

```python
legend(*args, **kwargs)
```

- Place a legend on the current axes at location `loc`
- Labels are a sequence of strings
- `loc` can be a string or an integer
Sample Legend Commands

# make a legend with existing lines
legend()

# automatically generate the legend from labels
legend( ('label1', 'label2', 'label3') )

# Make a legend for a list of lines and labels
legend( (line1, line2, line3), ('label1', 'label2', 'label3') )

# make a legend at a given location, using a location argument
legend( ('label1', 'label2', 'label3'), loc='upper left')
legend( (line1, line2, line3), ('label1', 'label2', 'label3'), loc=2)
A Graph with Legend

Minimum Message Length

- Model length
- Data length
- Total message length

Message length

Model complexity
Another Graph with Legend
Colorbar

You need to include the call: `colorbar()`
Contour Plot

# Make a contour plot of an array Z.
# The level values are chosen automatically
contour(Z)

# X, Y specify the (x, y) coordinates of the surface
contour(X, Y, Z)

# contour N automatically-chosen levels
contour(Z, N)
contour(X, Y, Z, N)

# draw contour lines at the values specified in sequence V
contour(Z, V)
contour(X, Y, Z, V)

# fill the (len(V)-1) regions between the values in V
contourf(..., V)
A Graph with Contour Plot
Another Graph with Contour Plot

A spiral!
The mplot3d Module

- The mplot3d toolkit adds simple 3d plotting capabilities to Matplotlib by supplying an axis object that can create a 2d projection of a 3d scene.
- It produces a list of 2d lines and patches that are drawn by the normal Matplotlib code.
- The resulting graph will have the same look and feel as regular 2d plots.
- Provide the ability to rotate and zoom the 3d scene.
Matplotlib's 3D capabilities were added by incorporating:

```python
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

fig = plt.figure()
ax = Axes3D(fig)
x = ...
y = ...
z = ...
ax.TYPE_of_Plot(x, y, z, ...)
```
Example of 2D Function

Assume that we want to plot the function:

\[ z = \sin (\sqrt{x^2 + y^2}) \]

\[ -5 \leq x, y \leq 5 \]
Code for Plotting 2D Function

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

from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = Axes3D(fig)
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.jet)
plt.show()
```
3D Surface Demo
```python
from pylab import imread, imshow

a = imread('myImage.png')
imshow(a)
```
Plotting Geographical Data Using Basemap

- Matplotlib toolkit
- Collection of application-specific functions that extends Matplotlib functionalities
- Provides an efficient way to draw Matplotlib plots over real world maps
- Useful for scientists such as oceanographers and meteorologists.
Defining a Basemap Object

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

# Lambert Conformal map of USA lower 48 states
m = Basemap(llcrnrlon=-119,
             llcrnrlat=22,
             urcrnrlon=-64,
             urcrnrlat=49,
             projection='lcc',
             lat_1=33,
             lat_2=45,
             lon_0=-95,
             resolution='h',
             area_thresh=10000)
```
Arguments for Defining a Basemap Object

projection: Type of map projection used
lat_1: First standard parallel for lambert conformal, albers equal area and equidistant conic
lat_2: Second standard parallel for lambert conformal, albers equal area and equidistant conic.
lon_0: Central meridian (x-axis origin) - used by all projections
llcrnrlon: Longitude of lower-left corner of the desired map domain
llcrnrlat: Latitude of lower-left corner of the desired map domain
urcrnrlon: Longitude of upper-right corner of the desired map domain
urcrnrlat: Latitude of upper-right corner of the desired map domain
resolution: Specifies what the resolution is of the features added to the map (such as coast lines, borders, and so on), here we have chosen high resolution (h), but crude, low, and intermediate are also available.
area_thresh: Specifies what the minimum size is for a feature to be plotted.
In this case, only features bigger than 10,000 square kilometer
Defining Borders

```python
# draw the coastlines of continental area
m.drawcoastlines()

# draw country boundaries
m.drawcountries(linewidth=2)

# draw states boundaries (America only)
m.drawstates()
```
Coloring the Map

```python
# fill the background (the oceans)
m.drawmapboundary(fill_color='aqua')

# fill the continental area
# we color the lakes like the oceans
m.fillcontinents(color='coral', lake_color='aqua')
```
Drawing Parallels and Meridians

# We draw a 20 degrees graticule of parallels and
# meridians for the map.
# Note how the labels argument controls the
# positions where the graticules are labeled
# labels=[left, right, top, bottom]

m.drawparallels(np.arange(25, 65, 20), labels=[1, 0, 0, 0])
m.drawmeridians(np.arange(-120, -40, 20), labels=[0, 0, 0, 1])
US Map
Using Satellite Background

```python
# display blue marble image (from NASA)
# as map background
m.bluemarble()
```
US Map with Satellite Background
Cities over a Map

cities = ['London', 'New York', 'Madrid', 'Cairo', 'Moscow', 'Delhi', 'Dakar']

lat = [51.507778, 40.716667, 40.4, 30.058, 55.751667, 28.61, 14.692778]

lon = [-0.128056, -74, -3.683333, 31.229, 37.617778, 77.23, -17.446667]

m = Basemap(projection='ortho', lat_0=45, lon_0=10)
m.drawmapboundary()
m.drawcoastlines()
m.fillcontinents()

x, y = m(lon, lat)
plt.plot(x, y, 'ro')

for city, xc, yc in zip(cities, x, y):
    plt.text(xc+250000, yc-150000, city, bbox=dict(facecolor='yellow', alpha=0.5))
Graph of Cities over a Map
# make up some data on a regular lat/lon grid.

```python
nlats = 73; nlons = 145; delta = 2.*np.pi/(nlons-1)
lats = (0.5*np.pi-delta*np.indices((nlats,nlons))[0,:,:])
lons = (delta*np.indices((nlats,nlons))[1,:,:])
wave = 0.75*(np.sin(2.*lats)**8*np.cos(4.*lons))
mean = 0.5*np.cos(2.*lats)*((np.sin(2.*lats))**2 + 2.)
```

# compute native map projection coordinates of 
# lat/lon grid.
```python
x, y = m(lons*180./np.pi, lats*180./np.pi)
```

# contour data over the map.
```python
CS = m.contour(x,y,wave+mean,15,linewidths=1.5)
```
Graph of Data over a Map
Graph of Data over a Map with Satellite Background
netCDF4
Useful Links for netCDF4

- **Introduction**
  
  [http://netcdf4-python.googlecode.com/svn/trunk/docs/netCDF4-module.html](http://netcdf4-python.googlecode.com/svn/trunk/docs/netCDF4-module.html)
What is netCDF4?

- Python interface to the netCDF version 4 library.
- Can read and write files in both the new netCDF 4 and the netCDF 3 format.
- Can create files that are readable by HDF5 utilities.
- Relies on NumPy arrays.
Opening a netCDF File

```python
from netCDF4 import Dataset
ncFid = Dataset(ncFileName, mode=modeType, format=fileFormat)
ncFid.close()
```

**modeType can be:** `'w'`, `'r+'`, `'r'`, or `'a'`

**fileFormat can be:** `'NETCDF3_CLASSIC'`, `'NETCDF3_64BIT'`, `'NETCDF4_CLASSIC'`, `'NETCDF4'`
Creating Dimensions in a netCDF File

```python
1. time = ncFid.createDimension('time', None)
2. lev = ncFid.createDimension('lev', 72)
3. lat = ncFid.createDimension('lat', 91)
4. lon = ncFid.createDimension('lon', 144)

print ncFid.dimensions
```
Creating Variables in a netCDF File

```python
1 times = ncFid.createVariable('time', 'f8', ('time',))
2 levels = ncFid.createVariable('lev', 'i4', ('lev',))
3 latitudes = ncFid.createVariable('lat', 'f4', ('lat',))
4 longitudes = ncFid.createVariable('lon', 'f4', ('lon',))
5
6 temp = ncFid.createVariable('temp', 'f4',
    ('time', 'lev', 'lat', 'lon'))
```
Adding Variable Attributes in a netCDF File

```python
ncFid.description = 'Sample netCDF file'
ncFid.history = 'Created for GSFC on March 25, 2013'
ncFid.source = 'netCDF4 python tutorial'
latitudes.units = 'degrees north'
longitudes.units = 'degrees east'
levels.units = 'hPa'
temp.units = 'K'
times.units = 'hours since 0001-01-01 00:00:00.0'
times.calendar = 'gregorian'
```
import numpy

latitudes[:], = numpy.arange(-90, 91, 2.0)
longitudes[:], = numpy.arange(-180, 180, 2.5)
levels[:], = numpy.arange(0, 72, 1)

from numpy.random import uniform
temp[0:5,:,:,:,:] = uniform(
    size=(5, levels.size, latitudes.size, longitudes.size))
Reading Data from a netCDF File

```python
ncFid = Dataset('myFile.nc4', mode='r')
time = ncFid.variables['time'][:]
lev = ncFid.variables['lev'][:]
lat = ncFid.variables['lat'][:]
lon = ncFid.variables['lon'][:]
temp = ncFid.variables['temp'][:]
```
H5py
Useful Links for H5py

- Quick Start Guide
  http://www.h5py.org/docs/intro/quick.html
What is H5py?

- A Python-HDF5 interface
- Allows interaction with files, groups and datasets using traditional Python and NumPy syntax.
- No need to know anything about HDF5 library.
- The files it manipulates are "plain-vanilla" HDF5 files.
Opening a HDF5 File

```python
import h5py
hFid = h5py.File('myfile.h5', modeType) 
```
```python
hFid.close()
```

**modeType can be:** `'w', 'w-', 'r+', 'r', or 'a'
Creating Dimensions in a HDF5 File

```python
lat = numpy.arange(-90, 91, 2.0)
dset = hFid.require_dataset('lat', shape=lat.shape)
dset[...] = lat
dset.attrs['name'] = 'latitude'
dset.attrs['units'] = 'degrees north'

lon = numpy.arange(-180, 180, 2.5)
dset = hFid.require_dataset('lon', shape=lon.shape)
dset[...] = lon
dset.attrs['name'] = 'longitude'
dset.attrs['units'] = 'degrees east'

lev = numpy.arange(0, 72, 1)
dset = hFid.require_dataset('lev', shape=lev.shape)
dset[...] = lev
dset.attrs['name'] = 'vertical levels'
dset.attrs['units'] = 'hPa'
```
Creating Variables in a HDF5 File

```python
from numpy.random import uniform
arr = np.zeros((5, lev.size, lat.size, lon.size))
arr[0:5,:,:,:] = uniform(
    size=(5, lev.size, lat.size, lon.size))
dset = hFid.require_dataset('temp', shape=arr.shape)
dset[...] = arr
dset.attrs['name'] = 'temperature'
dset.attrs['units'] = 'K'
```
Creating Groups in a HDF5 File

```python
gpData2D = hFid.create_group('2D_Data')
sgpLand = gpData2D.create_group('2D_Land')
sgpSea = gpData2D.create_group('2D_Sea')
gpData3D = hFid.create_group('3D_Data')
```
Writing Data in a Group

temp = gpData3D.create_dataset('temp', data=arr)
temp.attrs['name'] = 'temperature'
temp.attrs['units'] = 'K'
Reading Data from a HDF5 File

```python
hFid = h5py.File('myFile.h5', 'r')
lev = hFid['lev'].value
lat = hFid['lat'].value
lon = hFid['lon'].value
time = hFid['time'].value

temp1 = hFid['temp'].value

temp2 = hFid['3D_Data']['temp'].value

hFid.close()
```
Visualizing Gridded Data
Goals

- Access a netCDF file
- Retrieve data from the netCDF file
- Manipulate and plot the data
Code for Reading SLP

```python
ncFid = Dataset(fileName, mode='r')

lat = ncFid.variables['lat'][:]
lon = ncFid.variables['lon'][:]

slp = 0.01*ncFid.variables['SLP'][:]

ncFid.close()

nlat = lat.size - 1
nlon = lon.size - 1

mySLP = slp[0,:,:]```

Visualizing Gridded Data

Code for Plotting

```python
fig = plt.figure(1, figsize=(15,8), dpi=75)
ax = fig.add_axes([0.05, 0.05, 0.9, 0.85])
m = Basemap(projection='mill',
    llcrnrlat=lat[0], urcrnrlat=lat[nlat],
    llcrnrlon=lon[0], urcrnrlon=lon[nlon])
m.drawcoastlines(linewidth=1.25)
m.fillcontinents(color='0.8')
m.drawparallels(np.arange(-80,81,20), labels=[1,1,0,0])
m.drawmeridians(np.arange(-180,180,60), labels=[0,0,0,1])
im = m.imshow(mySLP,
    interpolation='nearest',
    extent=[lon[0], lon[nlon], lat[0], lat[nlat]],
    cmap=plt.cm.jet)
plt.colorbar(orientation='horizontal', shrink=.8)
plt.title('Sea Level Pressure')
plt.savefig('fig_slp.png')
plt.show()
```
Plot of SLP
Visualizing Gridded Data

Define a Generic Function for Contour Plots

```python
def bmContourPlot(var, lats, lons, figName, figTitle):
    plt.figure()
    latLow = lats[0]; latHigh = lats[-1]
    lonLow = lons[0]; lonHigh = lons[-1]
    m = Basemap(projection='mill',
                llcrnrlat=latLow, urcrnrlat=latHigh,
                llcrnrlon=lonLow, urcrnrlon=lonHigh,
                resolution='c')
    m.drawcoastlines()
    m.drawparallels(np.arange(latLow, latHigh+1, 30.))
    m.drawmeridians(np.arange(lonLow, lonHigh+1, 60.))
    longrid, latgrid = np.meshgrid(lons, lats)
    x, y = m(longrid, latgrid)
    m.contour(x, y, var); m.contourf(x, y, var)
    plt.title(figTitle)
    plt.colorbar(shrink=.8)
    plt.savefig(figName + '.png')
    plt.show()
```

J. Kouatchou and H. Oloso (SSSO)
Visualizing Gridded Data

Code for Plotting Mean and Variance of Temperature at 500mb

time = ncFid.variables['time'][:]
lev = ncFid.variables['lev'][:]
lat = ncFid.variables['lat'][:]
lon = ncFid.variables['lon'][:]
T = ncFid.variables['T'][:]

level500 = 29  # level of interest
T500 = T[:,level500,:,:]  # time, lat, lon
T500mean = np.mean(T500,0)
T500var = np.var(T500,0)

bmContourPlot(T500mean, lat, lon, 'fig_TempMean', 'Spatial Temperature Mean')
bmContourPlot(T500var, lat, lon, 'fig_TempVariance', 'Spatial Temperature Variance')
Plot of the Mean of Temperature
Plot of the Variance of Temperature
Slicing the Data

Assume that we want to plot the data in prescribed latitude and longitude ranges.

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

import numpy as np

def sliceLatLon(lat, lon, (minLat, maxLat), (minLon, maxLon)):
    indexLat = np.nonzero((lat[:] >= minLat) & (lat[:] <= maxLat))[0]
    indexLon = np.nonzero((lon[:] >= minLon) & (lon[:] <= maxLon))[0]
    return indexLat, indexLon
```
Plot of the Mean of Temperature (Slice)
Plot of the Variance of Temperature (Slice)
References I

Hans Petter Langtangen.
*A Primer on Scientific Programming with Python.*

Johnny Wei-Bing Lin.
*A Hands-On Introduction to Using Python in the Atmospheric and Oceanic Sciences.*

Drew McCormack.
*Scientific Scripting with Python.*
2009.

Sandro Tosi.
*Matplotlib for Python Developers.*
2009.