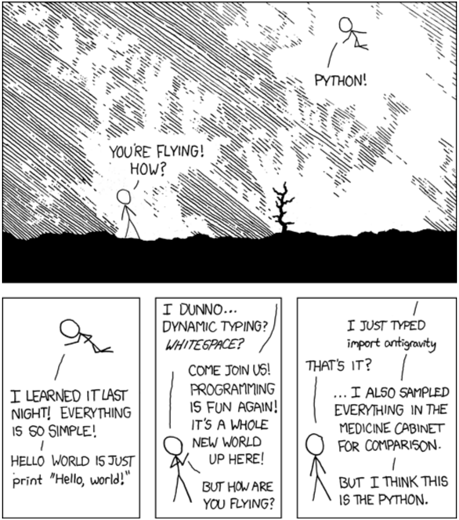
Python

Python makes you fly



Let's start

ipython vs python

ipython

- Note that you have an access to the OS shell
  - `ls`
  - `pwd`
  - reverse search: `ctrl + R`
- Automatic completion works !
  - use Tab
- Comments with #
- Great help
  - use ? or help
    - help pow
    - pow?

iPython II

History

- arrow up (shows previous command in history)
- arrow down (shows next command in history)
- po arrow up (shows previous command starting with po)
- history # (prints the complete recorded history)

Array basics

You need numpy

```
import numpy as np
```

Alternatively, you can use (less safe)

```
from numpy import *
```

Several possibilities how to create an array

- `x=np.zeros((20,30))`
  - Creates a 20x30 array of zeros
- `x=np.ones((20,30))`
- `x=np.arange(10)`
  - Creates integer arrays with sequential values
  - Starting with 0
- `x=np.arange(10.)`

Array basics II

- `x=np.arange(3,10)`
- `x=np.arange(3.,10.,1.5)`
- `x=np.array([3.,1.,7.])`
  - Creating arrays form literal arguments
- `x=np.array([[3.,1.][1.,7.]])`

Be careful about reals and integers !

```
x=3
y=4
x/y
```

```
x=3.
y=4
x/y
```

Array indexing starts from 0!

x,x[0],x[1]

Array types

Array numeric types

- The default for integer is usually 32-bit integers (in numpy called `int32`)
- The default for floats is 64-doubles (in numpy called `float64`)

How to find out the type of an array x with `dtype`

- `x=arange(4.)`
- `x.dtype`

Converting an array to a different type with `adtype()`

- `y=x.astype(float32)`

Array operations

- x.max()
- x.min()
- x.std()
- x.mean()
- x1=x.copy()
- x-=x.mean()
  - equivalent with `x=x-x.mean()`

- max(x)
- min(x)
- std(x)
- mean(x)
- x1=copy(x)
- x-=mean(x)
  - equivalent with `x=x-mean(x)`

First and last 10 entries of an array

```
x = np.random.random(100)
x[:10]
x[-100:]
x[2:5]
```

Multidimensional arrays

Creating an $N \times M$ array

```
N = 5
M = 3
x2 = np.zeros((N,M))
```

Size and shape of an array

```
size(x2)
shape(x2)
```

Reshaping of an array

```
x3 = np.arange(10).reshape(2,5)
```

Linear algebra

```
import numpy.linalg as la
```

An inverse of a matrix

```
x = np.array([[1,1],[1,-1]])
la.inv(x)
dot(x,la.inv(x))
```

Finding an eigenvalues and vectors

```
x = np.array([[1,2],[2,1]])
val, vec = la.eig(x)
```

Pyhton For statements

- ```
  for i in range(3):
      x=i*i
      print x
  ```

- - Indentation determines a block of code (such as for) cycle
  - In interactive mode starting a block causes the interpreter to prompt with ...(typing an empty line with enter terminated the block)
  - Tabs may be used for indentation, but their use is not recommended
  ```

Python If statements

- ```
  x=0
  if x==0:
      print "x equals 0"
  elif x==1:
      print "x equals 1"
  else:
      print "x equals something else than 0 or 1"
  ```
  ```

Plotting - handled with matplotlib module

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

Plotting random series of y values

```
y = np.random.random(50)
plt.figure()
plt.plot(x)
plt.show()
```

Plotting II

Plotting x against y values

```
t = np.arange(10)
y1 = np.random.random(10)
y2 = np.random.random(10)
plt.figure()
plt.plot(t,y1,'r–')
plt.plot(t,y2,'go-') plt.show()
```

Clear figure

```
plt.clf()
```

Close all figures

```
plt.close('all')
```

Other modules useful in Earth sciences

For seismologist

obspy module

- www.obspy.org

Geochemical systems - one reservoir

The concentration C_i of an element i must follow (in the simplest case)

$$\frac{\mathrm{d}C_i}{\mathrm{d}t} = -\frac{C_i}{\tau}$$

- Concentration in the box is homogeneous
- Initial condition $C_i(t=0) = C_0$

Ordinary differential equations (ODE)

We will need SciPy (Scientific tools for Python)

```
import scipy as sp
from scipy.integrate import odeint
```

The function `odeint` finds solution of equation

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}, t)$$

- With initial conditions $y(0) = y_0$
- Where $\mathbf{y}$ is a length N vector and $\mathbf{f}$ is a mapping from $\mathbb{R}^N$ to $\mathbb{R}^N$

Make your first Python script

```
touch box_model_single.py
```

```
#!/bin/python
import numpy as np
from scipy.integrate import odeint
import matplotlib as mpl
import matplotlib.pylab as plt
```

Parameters of the system

```
tau=1
y0=1    # initial conditions
```

ODE II

Define times when you want to find a solution

```
t = np.arange(0,10., 1.)
```

Right hand side of the equation

```
def func(y,t):
    return -y/tau
```

Solve the system

```
yy=odeint(func,y0,t) # solution of dC/dt=...
```

ODE III

We know analytical solution

```
time = np.arange(0,10.,0.1)
concentration=y0*np.exp(-time/tau) # analytical solution
```

Plotting

```
plt.figure()
plt.title("Concentration time evolution")
plt.ylabel("Concentration")
plt.xlabel("Time")
plt.plot(t,yy,'ro')
plt.plot(time,concentration)
```

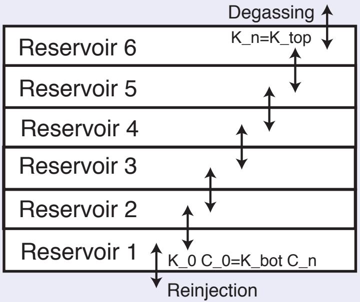
Run your script in iPython

```
execfile('box_model_single.py')
```

Box model - n reservoirs with concentrations C_i

Coltice at al. (GRL, 2000)

Transport of an inert gas in the mantle



The n equations to solve

$$\frac{\mathrm{d}C_i}{\mathrm{d}t} = \frac{v_z n}{L}(K_{i-1}C_{i-1} - K_i C_i)$$

- $K_i = 1 \Leftrightarrow i \neq 1 \text{ or n}$
- $K_0 C_0 = K_{bot} C_n = 0.2 C_n$
- $K_n = K_{top} = 10$
- $v_z = 1\,\mathrm{mm/y}$ (vertical velocity)
- n number of boxes
- L size of the domain

Box model with Python

```
touch box_model.py
```

```
#!/bin/ipython import numpy as np
from scipy.integrate import odeint
import matplotlib as mpl
import matplotlib.pylab as plt
```

Parameters of the system

```
vz=0.001    # m/year
ktop=10.0
kbot=0.2
nn=100    # number of boxes
LL=3E6    # size of mantle [m]
```

Box model with Python II

```
t = np.arange(0,1.8E9, 1E6) # times - min, max, step
y0=np.ones(nn) # array of initial concentration
ynew=np.zeros(nn)
```

```
def func(y,time):
    for ii in range(nn):
        if ii==0: # bottom
            ynew[0]=kbot*y[nn-1]-y[0]
        elif ii==nn-1: # top
            ynew[nn-1]=y[nn-2]-ktop*y[nn-1]
        else:
            ynew[ii]=y[ii-1]-y[ii]
    alpha=vz*LL/nn
    return alpha*ynew
```

```
yy=odeint(func,y0,t)
```

Plotting the results

```
zz=np.arange(0.,1.,1./nn)
zz=np.append(zz,1.)

ntime=size(t)-1
print 'plotting at time:', t[ntime]

plt.figure()
plt.title("Concentration profile")
plt.xlabel("Concentration")
plt.ylabel("Height")
plt.plot(np.append(yy[ntime],yy[ntime][nn-1]),zz,drawstyle='steps')
plt.axis([0,max(yy[ntime]),0,1])
plt.savefig('fig1.pdf')
```

Plotting the results II: Animation

```
files=[]
fig=plt.figure(figsize=(5,5))
ax=fig.add_subplot(111)
jj=0
for ii in range(0,size(t),size(t)/20): # 20 frames
   jj=jj+1
   ax.cla()
   ax.plot(np.append(yy[ii],yy[ii][nn-1]),zz,drawstyle='steps')
   fname = 'fig%03d.png'%jj    print 'Saving frame', fname
   fig.savefig(fname)
   files.append(fname)
```

```
os.system("mencoder 'mf:_tmp*.png' -mf type=png:fps=10 /
-ovc lavc -lavcopts vcodec=wmv2 -oac copy -o animation.mpg")
```

```
mplayer
```