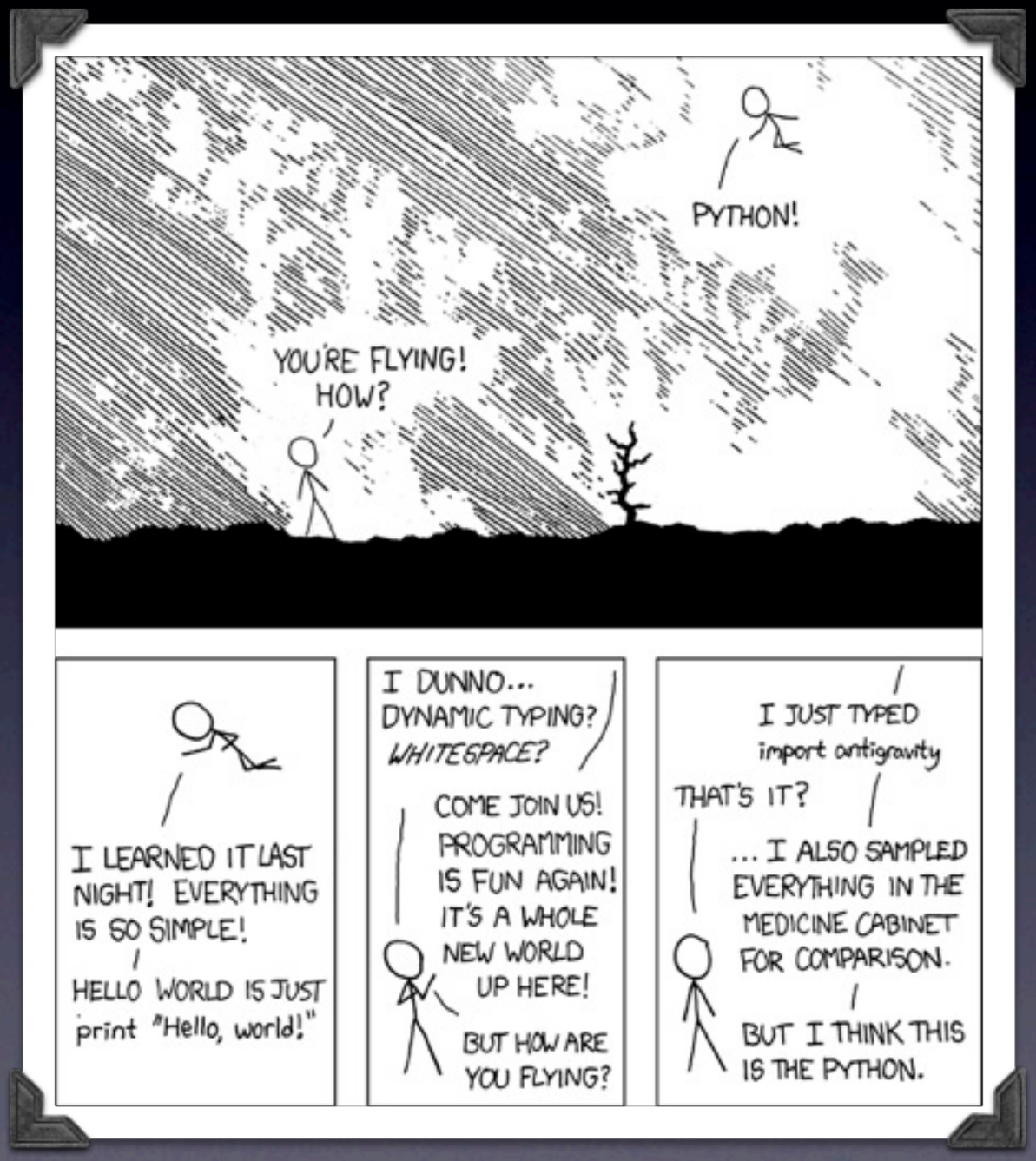
Python for Beginners

Simon Metson
s.metson@bris.ac.uk

Outline

- Introduction to python
  - from basic types to classes
- Some nice libraries
  - Random Numbers
  - Numpy
  - Matplotlib



http://xkcd.com/353/

This Course Will

- Introduce you to python
  - Running scripts, writing classes etc.
- Introduce you to some useful libraries for writing your own scripts and programs
  - Overview of plotting with MATPLOTLIB
  - Point you in the direction of all the wonderful documentation and third party resources
    - Google is always your friend!!
  - http://cern.ch/metson/PythonBeginners

Introduction to python

- Lets start with a show of hands for languages
  - Perl - http://pleac.sourceforge.net/pleac_python/index.html
  - Shell scripting (bash, tcsh etc)
  - Java
  - Fortran
  - C++
  - .Net
  - Lisp
  - Matlab
  - Others

About Python

- Written by a man named Guido
- Object Oriented, Interpreted
- Good for "Agile" development
- Latest release on HPC is 2.6
- Available for all major Operating systems

  "Python runs on Windows, Linux/Unix, Mac OS X, OS/2, Amiga, Palm Handhelds, and Nokia mobile phones. Python has also been ported to the Java and .NET virtual machines."

- http://python.org/

Baby steps

Starting the Interpreter

- Log in to blue crystal
- Type “module add python-2.6”
- Type “python”
- At the “>>>” prompt type help(‘modules’) to see a list of installed python libraries

```
Last login: Tue Feb  2 12:23:30 2010 from plague.phy.bris.ac.uk
 --- ---
|  |-|  |  Welcome to                          Based on Scientific Linux 4.3
 -(   )-   ClusterVisionOS 2.1                 ClusterVision ID: #50197
|  |-|  |
 --- ---

Use the following commands to adjust your environment:

'module avail'                - show available modules
'module add <module_name>'    - adds it to your environment

You should add modules you use regularly (eg. mpich) in your startup file.

Modules that add all default modules for a specific interconnect/compiler at once
are called e.g.'default-infiniband-pgi', 'default-infiniband-intel'.
-------------------------------------------------------------------------------
Use the command showquota to see your current disk use, quota etc.

Also see the file /exports/gpfs/Diskuse to see how much free disk
space there is, and how much space users/groups are using.
-------------------------------------------------------------------------------
Please note user data stored on the system is not backed up and it is therefore
your responsibility to ensure that you have adequate back up.
-------------------------------------------------------------------------------

[phxsm@bluecrystal2 ~]$ module add python-2.6
[phxsm@bluecrystal2 ~]$ python
Python 2.6 (r26:66714, Nov 12 2008, 10:18:00)
[GCC 3.4.6 20060404 (Red Hat 3.4.6-8)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> help('modules')

Please wait a moment while I gather a list of all available modules...
```

Lost?

- Probably the best thing about python is it's extensive documentation
- http://docs.python.org/
- Also the help() function, e.g help(set)

```
Help on class set in module __builtin__:

class set(object)
 |  set(iterable) --> set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(...)
 |      x.__and__(y) <==> x&y
 |
 |  __cmp__(...)
 |      x.__cmp__(y) <==> cmp(x,y)
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(...)
 |      x.__eq__(y) <==> x==y
```

Python as a Calculator

- Once in the interpreter you can write simple programs
- Once happy with these programs you can save them to a script
  - Of course, you can just write scripts...

```
[phxsm@bluecrystal1 ~]$ python
Python 2.5.2 (r252:60911, Mar 31 2008, 09:54:52)
[GCC 3.4.6 20060404 (Red Hat 3.4.6-8)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello"
Hello
>>> print 5 + 9
14
>>> five = 5
>>> neuf = 9
>>> print five + neuf
14
>>>
```

Running a Script

- Simply type

  python myscript.py

  at the shell prompt

- or make it executable

The Script

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

print "hello world"
```

The Output

```
[phxsm@bluecrystal1 ~]$ python myscript.py
hello world
[phxsm@bluecrystal1 ~]$ chmod u+x myscript.py
[phxsm@bluecrystal1 ~]$ ./myscript.py
hello world
[phxsm@bluecrystal1 ~]$
```

Nuts and Bolts

Syntax

- Python uses white space to identify code blocks

This will run...

```
if sky = ‘blue’:
    birds.sing()
elif sky = ‘black’:
    birds.sleep()
else:
    pass  #do nothing
```

...this won’t

```
if sky = ‘blue’:
birds.sing()
elif sky = ‘black’:
 birds.sleep()
else:
  pass  #do nothing
```

Control statements

- The usual suspects:
  - if/elif/else
  - for
  - while

```
if sky = ‘blue’:
    birds.sing()
elif sky = ‘black’:
    birds.sleep()
else:
    pass  #do nothing
```

```
for i in range(1,5):
    print i
```

```
i = 5
while i >= 0:
    print i
    i = i - 1
```

```
[phxsm@bluecrystal1 ~]$ python
Python 2.3.4 (#1, Dec 10 2007, 15:03:10)
[GCC 3.4.4 20050721 (Red Hat 3.4.4-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(1,5):
...     print i
...
1
2
3
4
>>> i = 5
>>> while i >= 0:
...     print i
...     i = i - 1
...
5
4
3
2
1
0
>>>
```

Control statements

```
[phxsm@bluecrystal1 ~]$ python
Python 2.3.4 (#1, Dec 10 2007, 15:03:10)
[GCC 3.4.4 20050721 (Red Hat 3.4.4-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(1,5):
...     print i
...
1
2
3
4
>>> i = 5
>>> while i >= 0:
...     print i
...     i = i - 1
...
5
4
3
2
1
0
>>>
```

Base Types

- Strings, integers, floats, booleans complex numbers
  - Python also has unicode strings
- Python is dynamically typed
- Python is “duck typed”

```
[phxsm@bluecrystal2 ~]$ python
Python 2.3.4 (#1, Dec 10 2007, 15:03:10)
[GCC 3.4.4 20050721 (Red Hat 3.4.4-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> myVariable = 123
>>> myVariable
123
>>> myVariable = "Simon"
>>> print myVariable
Simon
>>> myVariable
'Simon'
>>> myVariable = True
>>> myVariable
True
>>> myVariable = False
>>> myVariable
False
>>> myVariable = 123.456
>>> myVariable
123.456
>>> myVariable = 5 / 2
>>> myVariable
2
>>> myVariable = 5. / 2
>>> myVariable
2.5
```

```
>>> myVariable = 'Metson'
>>> myVariable
'Metson'
>>> myVariable = 2 + 0.5j
>>> myVariable
(2+0.5j)
>>> myVariable.real
2.0
>>> myVariable.imag
0.5
>>> myVariable.conjugate()
(2-0.5j)
>>> myVariable = 'Metson'
>>> len(myVariable)
6
```

String manipulation

- You're pretty much guaranteed to manipulate strings at some point, even if only to print out debug messages
- I seem to use split(), strip(), replace(), join() a lot
- http://docs.python.org/library/stdtypes.html#string-methods

Strings example

```
>>> my_string = "hello %s,\n\t it's nice to be here"
>>> my_string
"hello %s,\n\t it's nice to be here"
>>> print my_string
hello %s,
	 it's nice to be here
>>> print my_string % 'everyone'
hello everyone,
	 it's nice to be here
>>> my_list = my_string.split()
>>> my_list
['hello', '%s,', "it's", 'nice', 'to', 'be', 'here']
>>> for i in my_string.split('\n'):
...     print i.strip()
...
hello %s,
it's nice to be here
```

Simple grouping types

Lists

```
>>> aList = [1, 2, "three", 'IV']
>>> aList
[1, 2, 'three', 'IV']
>>> anotherList = [5,6,7]
>>> aList.append(anotherList)
>>> aList
[1, 2, 'three', 'IV', [5, 6, 7]]
>>> yetAnotherList = [8, 9, 10]
>>> aList.extend(yetAnotherList)
>>> aList
[1, 2, 'three', 'IV', [5, 6, 7], 8, 9, 10]
>>> len(aList)
8
>>> aList[4]
[5, 6, 7]
>>>
```

Tuple

```
>>> myTuple = (1, 2, "three", "IV")
>>> myTuple[0]
1
>>> myTuple[2]
'three'
>>> len(myTuple)
4
>>> myTuple[5] = "V"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

- Lists are mutable (you can change their contents)
- Tuple's are not
- You can easily iterate over contents of both (for i in myList...)

List slicing

- You can access elements in a list (start counting from 0)
- You can slice lists
- You can look at ranges in a list

```
>>> my_list = [1, 2, 3, 4, 5, 6]
>>> my_list
[1, 2, 3, 4, 5, 6]
>>> my_list[2]
3
>>> my_list[:4]
[1, 2, 3, 4]
>>> my_list[4:]
[5, 6]
>>> my_list[-1]
6
>>> my_list[-3:]
[4, 5, 6]
>>> my_list[2:4]
[3, 4]
```

Grouping types

Sets

```
>>> myset = set()
>>> myset.add("hello there")
>>> print myset
set(['hello there'])
>>> myotherset = set(["hello there"])
>>> myotherset.add("isn't this fun")
>>> myotherset - myset
set(["isn't this fun"])
>>> myotherset | myset
set(["isn't this fun", 'hello there'])
>>> myotherset & myset
set(['hello there'])
>>> myotherset ^ myset
set(["isn't this fun"])
```

Dictionaries

```
>>> theDictionary = {}
>>>
>>> print theDictionary
{}
>>>
>>> theDictionary['python'] = "any of several Old World boa
, often growing to a length of more than 20 ft. (6 m): the
red."
>>>
>>> theDictionary['dictionary'] = 'a book containing a sele
 arranged alphabetically, giving information about their m
ected forms, etc., expressed in either the same or another
>>>
>>> print theDictionary
{'python': 'any of several Old World boa constrictors of th
a length of more than 20 ft. (6 m): the Indian python, Pyth
: 'a book containing a selection of the words of a language
 information about their meanings, pronunciations, etymolog
either the same or another language'}
>>> print theDictionary['python']
any of several Old World boa constrictors of the subfamily
more than 20 ft. (6 m): the Indian python, Python molurus,
```

- Dictionaries are very useful, keys and values can be retrieved as lists
- Sets are great for maintaining unique lists and fast difference operations, and can be easily cast to a list: list(myset)
- More on sets: http://docs.python.org/library/sets.html

Exercise 1

- Start the interpreter and do a few simple sums
- Make a list of numbers, iterate through them, summing up even numbers
- Make a dictionary containing a list, a string and a complex number
- Make a set, try adding it to the dictionary

More interesting

Random Numbers

- Random numbers are used a lot in scientific computing, easy to use in Python

- Can produce random numbers by different algorithms/distributions

- http://docs.python.org/library/random.html

```
>>> from random import randint
>>> randint(0,10)
7
>>> randint(0,10)
9
>>> randint(0,10)
4
>>> from random import random
>>> random()
0.07676431985380927 1
>>> random()
0.20544223193230737
>>> from random import gauss
>>> help(gauss)

>>> gauss(2, 5)
-0.9907839929949065
>>> gauss(2, 5)
7.0730487031090412
>>> gauss(2, 5)
3.3587999617400293
>>> gauss(2, 5)
18.495205345051708
```

Functions/methods

- Functions/methods are identified by the keyword “def”, for instance

```
def hello():
        print “hello world”
```

- Same syntax for a class method, just indented more
- Can define functions in a script or the interpreter

Example

```
>>> def print_square(number):
...     print number * number
...     return 0
...
...
>>> print_square(10)
100
0
```

Aside: functions as objects

- Every thing in Python is an object, even a function
- A function takes objects as its arguments, which means it can take another function as an argument
  - Handy when writing integration/differentiation functions for instance
- A dictionary can hold an object as a value, which means you can make a dictionary of functions
- This is super awesome when you want to programatically decide what function to run
  - I totally abuse it

Aside Example

```
>>> def square(number):
...     return number * number
...
>>> def cube(number):
...     return number * number * number
...
>>> functions = {}
>>> functions['sq'] = square
>>> functions['cb'] = cube
>>> print functions
{'cb': <function cube at 0xba4f0>, 'sq': <function square
at 0xba4b0>}
>>> functions['sq'](2)
4
>>> functions['cb'](2)
8
```

Interacting with externals

- You can use python to run other programs and read their output
  - http://docs.python.org/library/subprocess.html
- You can read from the environment
  - os.environ['HOME']
  - http://docs.python.org/library/os.html

Example

```
>>> import subprocess
>>> import os
>>> def run(command):
...     proc = subprocess.Popen(
...             ["/bin/bash"], shell=True, cwd=os.environ['PWD'],
...             stdout=subprocess.PIPE,
...             stderr=subprocess.PIPE,
...             stdin=subprocess.PIPE,
...             )
...     proc.stdin.write(command)
...     stdout, stderr =  proc.communicate()
...     rc = proc.returncode
...     return stdout, stderr, rc
...
>>> run ('ls')
('COMP\nCouchapp-0.5.2\nDesktop\nDocuments\nDownloads\nEVO Recorded Streams
\nIcon\r\nLibrary\nMovies\nMusic\nPictures\nPublic\nSites\nWT_CVS\nWT_Devel
\ngits\nobi\npylibs\nscripts\nscripts.tar.gz\nsqlnet.log\ntnsnames.ora
\ntodo.sqlite\nworkspace\nworkspace_erlide.log\nworkspace_erlide.log.lck\n',
'', 0)
```

Exercise 2

- Copy the “run” function from the example, use it to run a few linux commands
  - for example; ls, ls -l, touch file, ls -l file
- Write a function that manipulates its input (say cubes it) and returns that result
- using run(“ls -l”) work out the total file size of your working directory

Here comes the science bit

NumPy

- Numerical tools for Python
- Matlab users look at http://www.scipy.org/NumPy_for_Matlab_Users
- We'll do a few simple things with NumPy

Arrays

```
>>> from numpy import *
>>> my_array = array([[1,2,3], [4,5,6], [7,8,9]])
>>> my_array
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> my_array.shape
(3, 3)
>>> my_array.size
9
>>> transpose(my_array)
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
>>> my_array = array([[1 + 0j, 2+ 1j], [3 + 2j, 4+0j]])
>>> transpose(my_array).conj()
array([[ 1.-0.j,  3.-2.j],
       [ 2.-1.j,  4.-0.j]])
>>> my_array.conj()
array([[ 1.-0.j,  2.-1.j],
       [ 3.-2.j,  4.-0.j]])
>>>
```

Arrays

```
>>> my_data = array([[1,2,3],[4,5,6]])
>>> average(my_data)
3.5
>>> my_weights = array([[1,1,0.5],[0.5,0.25,0.125]])
>>> average(my_data, weights=my_weights)
2.5185185185185186
>>> average(my_data, weights=my_weights,returned=True)
(2.5185185185185186, 3.375)
```

Matplotlib

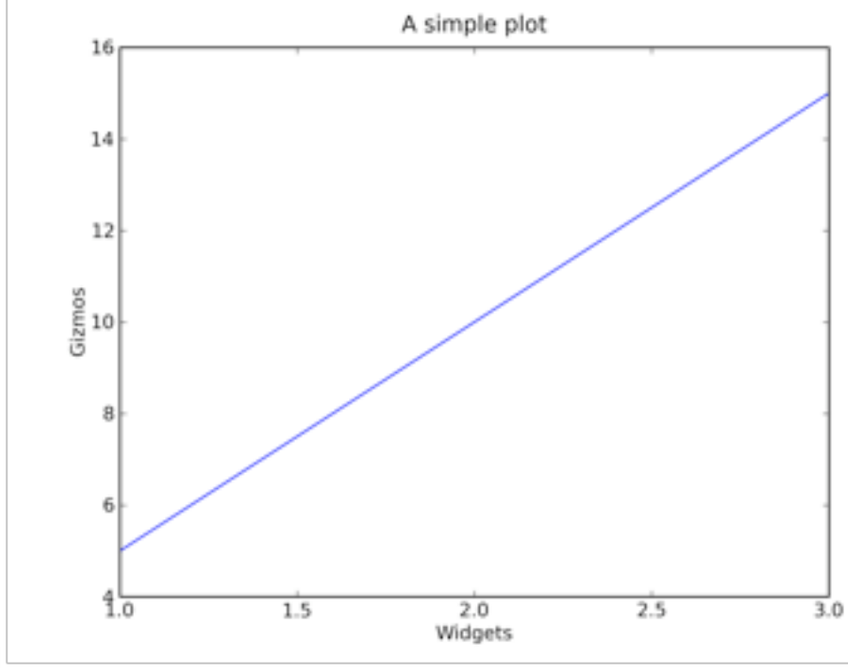
- Matplotlib is a plotting library, written in python very similar to MatLab (except its free!)
- Produces publication quality plots with minimal fuss
- Can easily integrate into your scripts
- http://matplotlib.sourceforge.net/

Using Matplotlib

- Making plots
  - Saving as image files
  - Interacting with plots
  - *I noticed last night that HPC doesn't have matplotlib installed for python 2.6, so use python2.5.2 (module add python-2.5.2)*

Making Plots

```
>>> import matplotlib
>>> matplotlib.use('Agg')
>>> import matplotlib.pyplot as plt
>>> from pylab import savefig
>>> plt.plot([1,2,3], [5,10,15])
[<matplotlib.lines.Line2D object at 0x2069570>]
>>> plt.ylabel('Gizmos')
<matplotlib.text.Text object at 0x2086eb0>
>>> plt.xlabel('Widgets')
<matplotlib.text.Text object at 0x20865f0>
>>> plt.title('A simple plot')
<matplotlib.text.Text object at 0x222e7f0>
>>> savefig('curves', dpi=300)
```



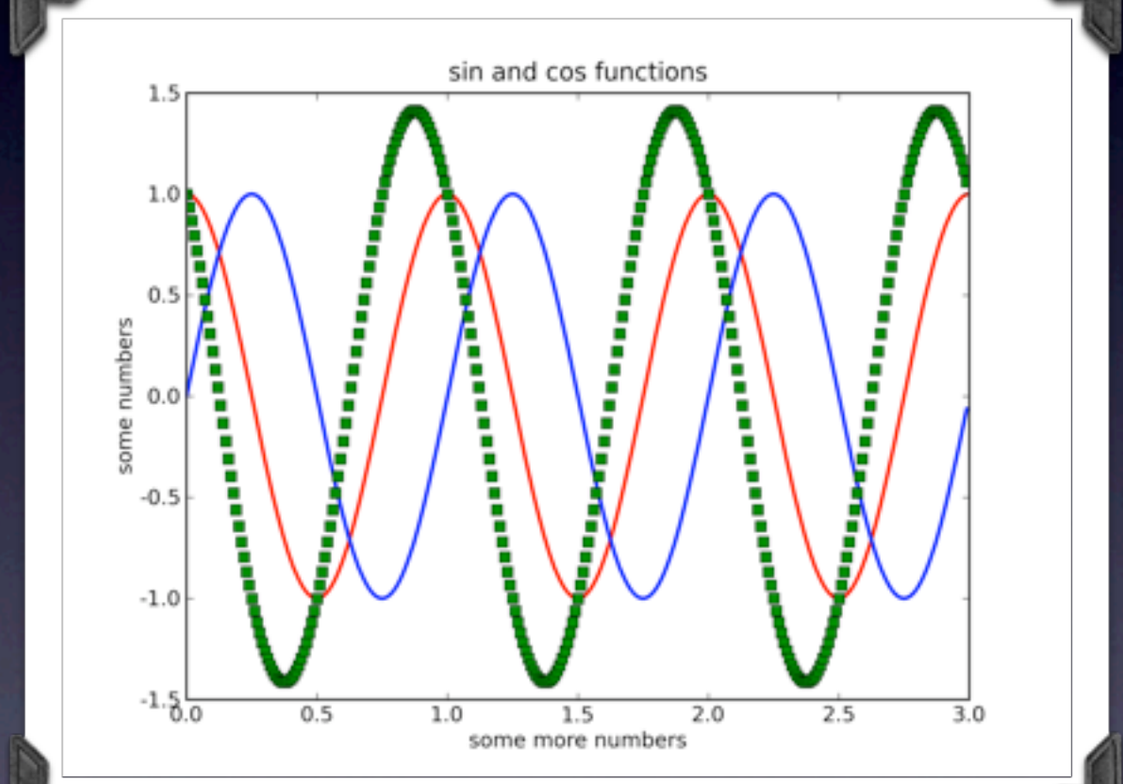
- The code on the left creates the plot on the right as curves.png in the directory where python was started

Interacting with plots

- Labelling axes
  - plot.ylabel(), plot.xlabel()
- Changing colours and markers
  - >>> plt.plot([1,2,3], [5,10,15], 'g*')
- Labelling plots
  - plot.title()

Interacting with plots

```
Python 2.5.2 (r252:60911, Mar 31 2008, 09:54:52)
[GCC 3.4.6 20060404 (Red Hat 3.4.6-8)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import matplotlib
>>> matplotlib.use('Agg')
>>> import matplotlib.pyplot as plt
>>> from numpy import  arange, pi, cos, sin
>>> from pylab import savefig
>>> t = arange(0.0, 3.0,0.01)
>>> print t[:4] # print first 4 elements of t
[ 0.    0.01  0.02  0.03]
>>> c = cos(2*pi*t)
>>> s = sin(2*pi*t)
>>> plt.ylabel('some numbers')
<matplotlib.text.Text instance at 0x2a9caaff80>
>>> plt.xlabel('some more numbers')
<matplotlib.text.Text instance at 0x2a9caaf368>
>>> plt.plot(t, c, 'r', lw=2)
[<matplotlib.lines.Line2D instance at 0x2a9cab2098>]
>>> plt.plot(t, s, 'b', lw=2)
[<matplotlib.lines.Line2D instance at 0x2a9cab2128>]
>>> plt.plot(t, c-s, 'gs', lw=2)
[<matplotlib.lines.Line2D instance at 0x2a9cab2200>]
>>> plt.ylim(-1.5, 1.5)
(-1.5, 1.5)
>>> plt.ylabel('some numbers')
<matplotlib.text.Text instance at 0x2a9caaff80>
>>> plt.xlabel('some more numbers')
<matplotlib.text.Text instance at 0x2a9caaf368>
>>> plt.title('sin and cos functions')
<matplotlib.text.Text instance at 0x2a9cab1cf8>
>>> savefig('curves', dpi = 300)
```



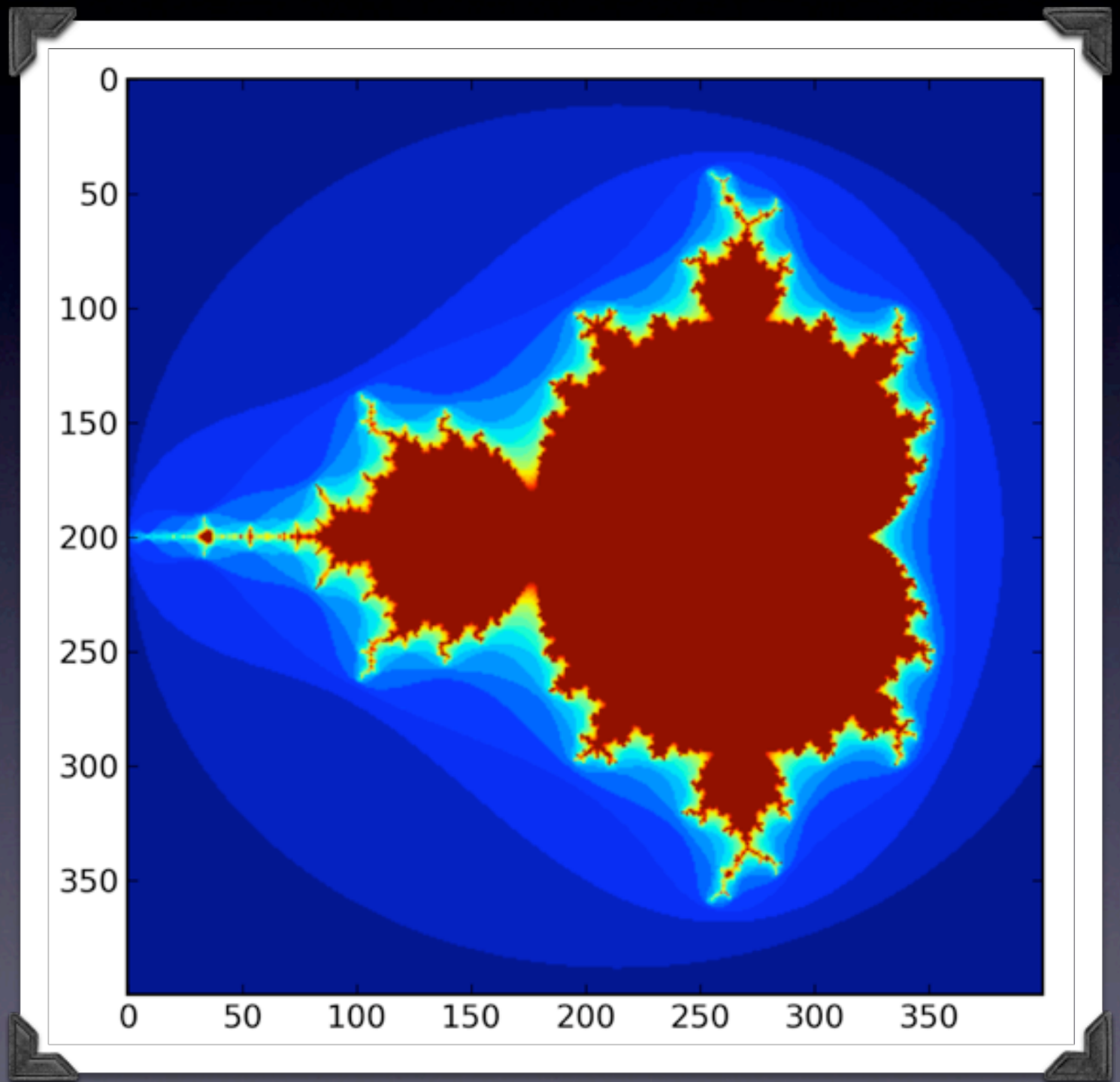
Types of plot

- Many!
  - Scatter, polar, bar chart
- Consult the manual for details
- http://matplotlib.sourceforge.net/gallery.html

Mandlebrot

```
>>> from numpy import *
>>> from matplotlib import use as use_backend
>>> use_backend('Agg')
>>> from pylab import savefig, imshow
>>> def mandelbrot( h,w, maxit=20 ):
...         '''Returns an image of the Mandelbrot fractal of size (h,w).
...         '''
...         y,x = ogrid[ -1.4:1.4:h*1j, -2:0.8:w*1j ]
...         c = x+y*1j
...         z = c
...         divtime = maxit + zeros(z.shape, dtype=int)
...         for i in xrange(maxit):
...                 z  = z**2 + c
...                 diverge = z*conj(z) > 2**2            # who is diverging
...                 div_now = diverge & (divtime==maxit)  # who is diverging now
...                 divtime[div_now] = i                  # note when
...                 z[diverge] = 2                        # avoid diverging too much
...         return divtime
...
>>>
>>> imshow(mandelbrot(400,400))
<matplotlib.image.AxesImage instance at 0x2a9cb24c68>
>>> savefig('mandy', dpi=300)
```

Mandlebrot



Exercise 3

- Calculate pi using a random number generator
- Complete the ex3-pi.py script
- Plot iterations at appropriate frequency using Matplotlib and save to a png file

Serious Python Fu

A Touch of Class

- Python is an Object Oriented language
- In python you are interacting with objects, even if you don't know it
- It is _very_ easy to make new objects
- If you don't understand OO just nod and smile

Defining a Class

- Defining a class uses, imaginatively, the key word “class”
- Methods are defined using “def” but have at least one argument “self”
- Inheritance (and multiple inheritance) is expressed by listing base classes in ()’s after the class name

Examples

```
class Person(object):
        def __init__(self, name=None, age=None, height=None):
                self.__name = name      //private instance variable
                self.__age = age        //private instance variable
                self.__height = height  //private instance variable

        def getName(self):
                return self.__name

        def getAge(self):
                return self.__age
```

```
class Person(dict):
        def __init__(self, name=None, age=None, height=None):
                self.setdefault("name", name)
                self.setdefault("age", age)
                self.setdefault("height", height)
```

This sort of thing ("getters") are generally frowned upon in OO*

Here we inherit from the dictionary object.

This is more python-ised, don't need to write the getter

* For a long article on why see: http://www.javaworld.com/javaworld/jw-09-2003/jw-0905-toolbox.html?page=1
- it's all about data abstraction and code maintainability...

Exercise 4

- Make a class that represents a person with a name, age and height
- Give the person class a grow(self, amount) method that increments the age by one and increases the height by amount

Other stuff

things I decided not to cover today...

Reading & Writing Files

- To read/write files you instantiate a File object and call it's read() or readlines() and write() methods.

```
>>> f=open('/tmp/workfile', 'w')
>>> f.write('This is a test\n')
>>> f.write('This is another test')
>>> f.close()
>>> f=open('/tmp/workfile', 'r')
>>> f.readlines()
['This is a test\n', 'This is another test']
>>> f.read()
''
>>> f.close()
>>> f=open('/tmp/workfile', 'r')
>>> f.read()
'This is a test\nThis is another test'
>>> f.read()
''
>>> f.seek(0)
>>> f.read()
'This is a test\nThis is another test'
>>> f.close()
```

- readlines() returns a list, one item per line, read() returns a single string
- seek() moves around in the file

Reading Files II

- If you want some structured data python gives you many options (XML, json, YAML)
- We'll look at ConfigParser and pickle

ConfigParser

- Write DOS like configuration files
- Standardised format, interchangable

```
>>> from ConfigParser import ConfigParser
>>>
>>> config = ConfigParser()
>>>
>>> config.add_section ("person")
>>> config.add_section ("ice cream")
>>>
>>> config.set ("person", "name", "Simon")
>>> config.set ("person", "age", "28")
>>> config.set ("person", "hair", "Long")
>>>
>>> config.set ("ice cream", "flavour", "mint choc chip")
>>> config.set ("ice cream", "quantity", "lots")
>>>
>>> config.write (file ('/tmp/configexample.ini', 'w'))
>>>
>>> config.read ('/tmp/configexample.ini')
>>> config.get("person", "name")
'Simon'
>>>
[phxsm@bluecrystal2 ~]$ head /tmp/configexample.ini
[person]
hair = Long
age = 28
name = Simon

[ice cream]
flavour = mint choc chip
quantity = lots
```

Pickle

- Serialise python objects to a file or string for transfer/later use
- Extremely simple - dump/load; dumps/loads

```
>>> import pickle
>>> dict = {'name': 'obi', 'type': 'pet'}
>>> pickle.dumps(dict)
"(dp0\nS'type'\np1\nS'pet'\np2\nsS'name'\np3\nS'obi'\np4\ns."
>>> f = open('obi', 'w')
>>> pickle.dump(dict, f)
>>> f.close()
>>> f = open('obi', 'r')
>>> loaded_dict = pickle.load(f)
>>> loaded_dict
{'type': 'pet', 'name': 'obi'}
```

Option Parsing

- Python has a nice way to give input to your scripts

```
from optparse import OptionParser

parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="write report to FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose", default=True,
                  help="don't print status messages to stdout")

(options, args) = parser.parse_args()
```

- Your input will be stored in a list called args and as options.filename, options.verbose
- http://www.python.org/doc/2.3.4/lib/module-optparse.html

Example Script

```
#!/usr/bin/env python
from optparse import OptionParser

def doOptions():
    parser = OptionParser()
    parser.add_option("-v", "--verbose", action="store_true",
dest="verbose", default=False, help="be verbose")
    return parser.parse_args()

def hello():
    return "Hello"

def main():
    opts, args = doOptions()
    if opts.verbose:
        print "This script prints 'Hello'..."
    print hello()
    if opts.verbose:
        print "... see"

if __name__ == "__main__":
    main()
```

```
[phxsm@bluecrystal2 ~]$ python hello.py
Hello
[phxsm@bluecrystal2 ~]$ python hello.py -v
This script prints 'Hello'...
Hello
... see
```