Scientific Python

By Edward Smith

19th September 2017

10am

Plan for Today

10:00 to 10:45 Review of yesterday

10:45 to 11:30 More on Numpy and plotting

11:30 to 12:00 Loading data from files

12:00 to 13:00 Lunch

13:00 to 13:45 Using Python as glue

13:45 to 14:30 More advanced plotting

14:30 to 15:30 A complete post-processing example

15:30 to 16:00 Best practice and summary

http://tinyurl.com/ichpcclass

Introduction

Pros and Cons of Python (vs e.g. MATLAB)

Pros

- Free and open-source
- Not just for scientific computing
- Great libraries (One of Google's languages)
- Clear, clever and well designed syntax
- Remote access (ssh)
- Great online documentation (stackoverflow!)

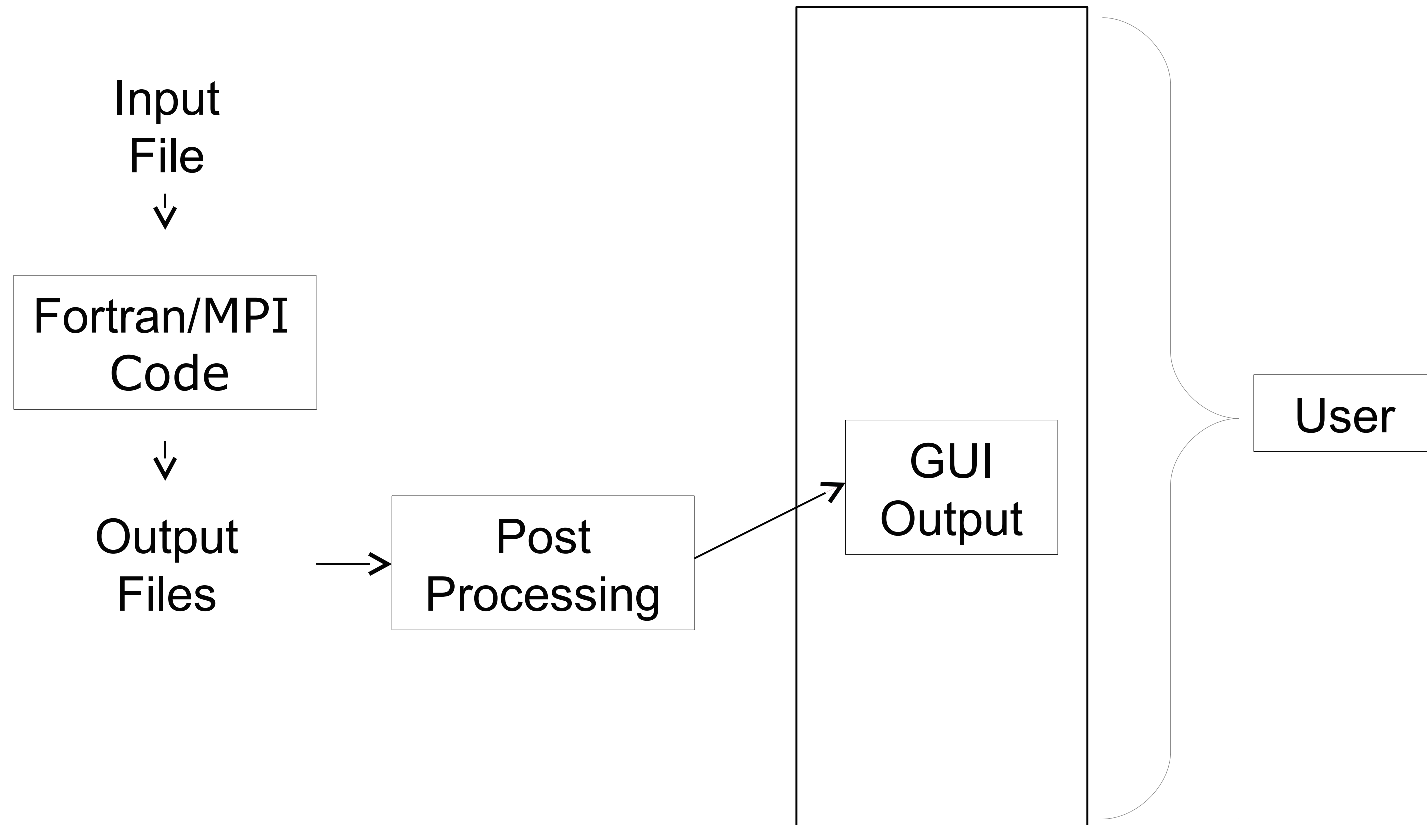
Cons

- No debugging GUI so less user friendly
- Syntax is different with some odd concepts
- No type checking can cause problems
- Not as many scientific toolboxes as MATLAB, inbuilt help not as good
- Slow compared to low level languages

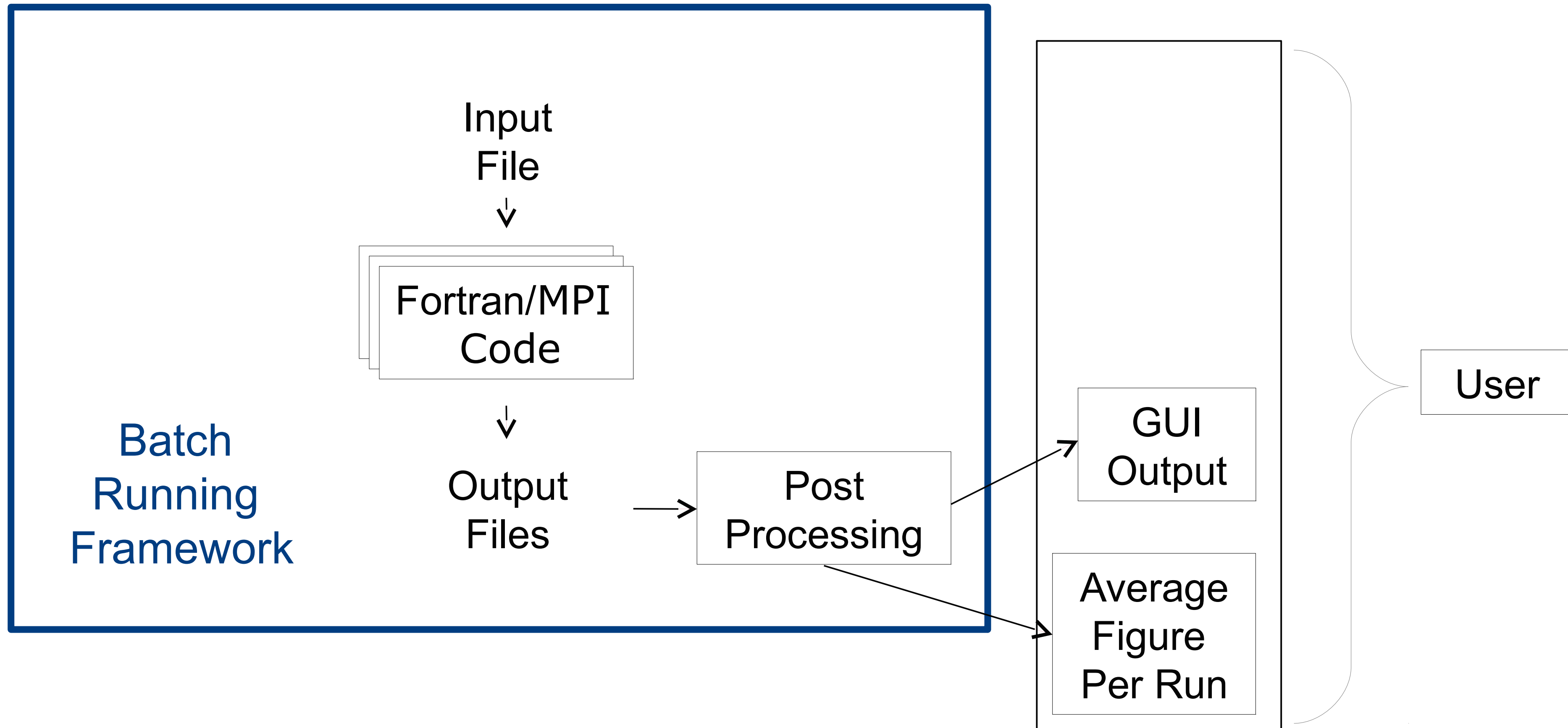
How I use Python in my Work

- Post processing framework
  - Low level data readers for a range of different data formats
  - Higher level field provide standard data manipulation to combine, average and prepare data to be plotted
- Visualiser Graphical User Interface
  - Read all possible field objects in a folder
  - Based on wxpython and inspired by MATLAB sliceomatic
- Batch running framework for compiled code
  - Simple syntax for systematic changes to input files
  - Specifiy resources for multiple jobs on desktop, CX1 or CX2
  - Copies everything needed for repeatability including source code, input files and initial state files

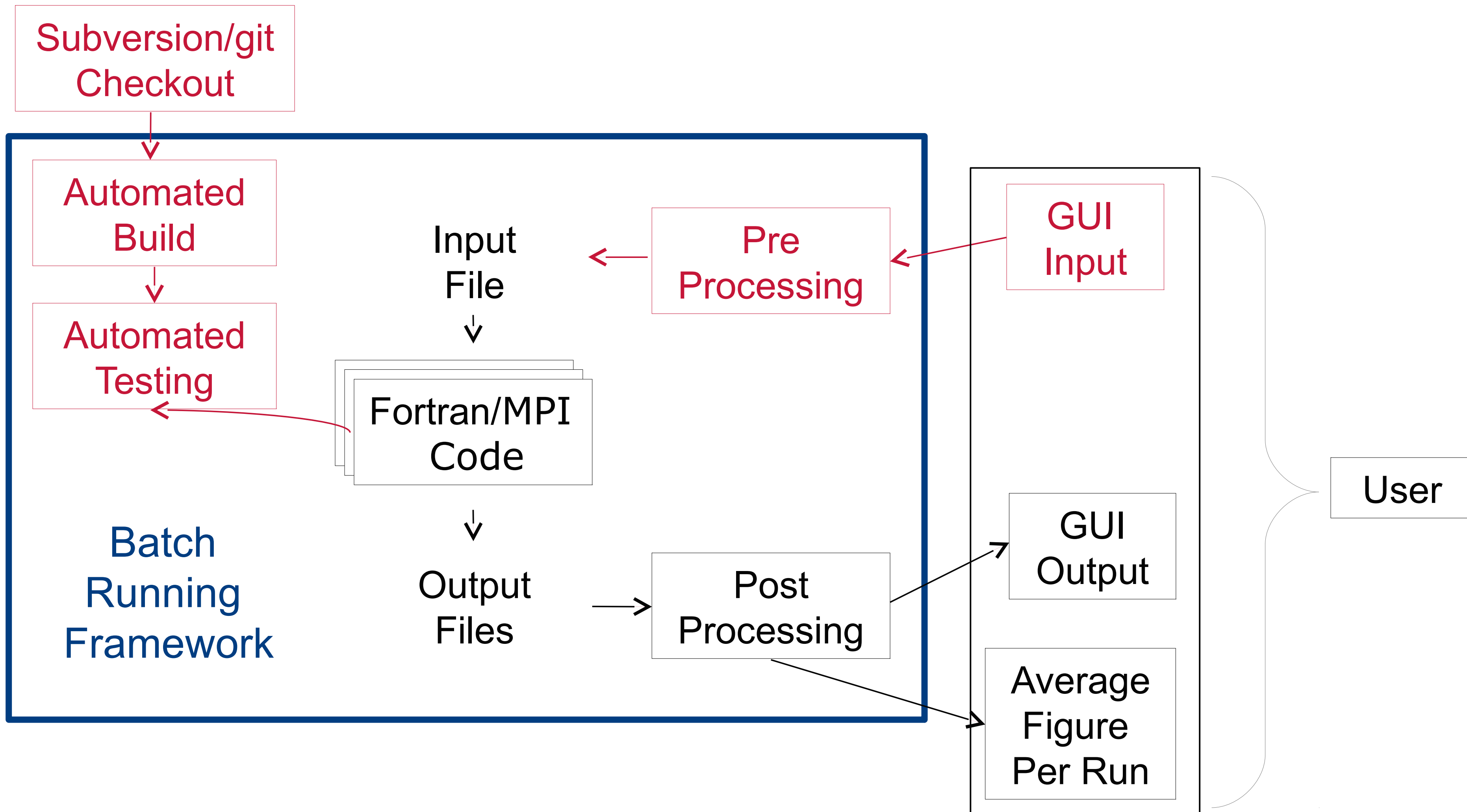
How I use Python in my Work



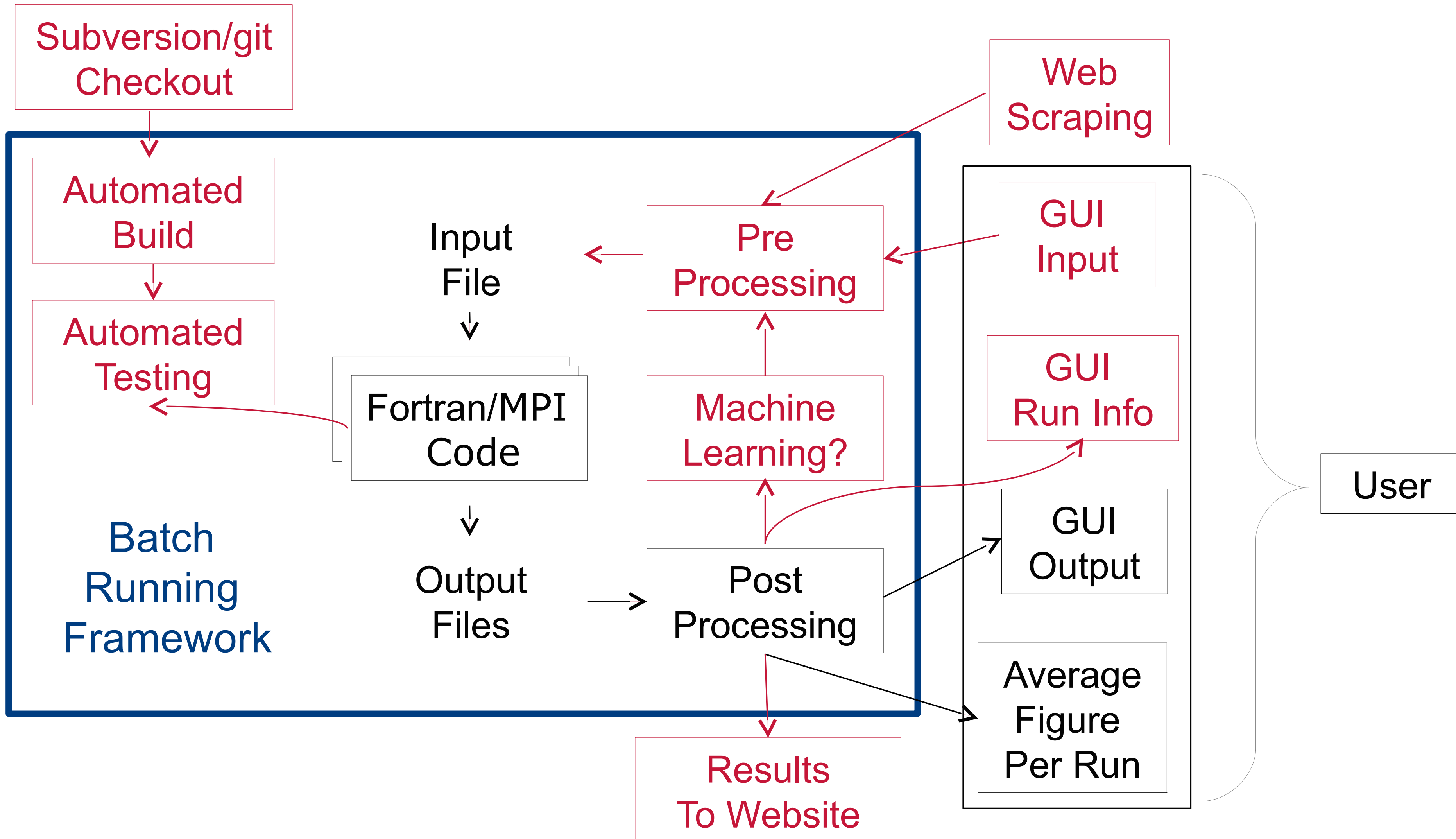
How I use Python in my Work



How I use Python in my Work



Possible Future Extensions



Aims for the course

- A focus on the strange or unique features of python as well as common sources of mistakes or confusion
- Help with the initial frustration of learning a new language
- Prevent subtle or undetected errors in later code
- Make sure the course is still useful to the wide range of background experiences

Review of Yesterday

What we covered yesterday

- Show how to use the command prompt to quickly learn Python
- Introduce a range of data types (Note everything is an object)

```
a = 3.141592653589        # Float
i = 3                     # Integer
s = "some string"         # String
l = [1,2,3]               # List, note square brackets tuple if ()
d = {"red":4, "blue":5}   # dictionary
x = np.array([1,2,3])     # Numpy array
```

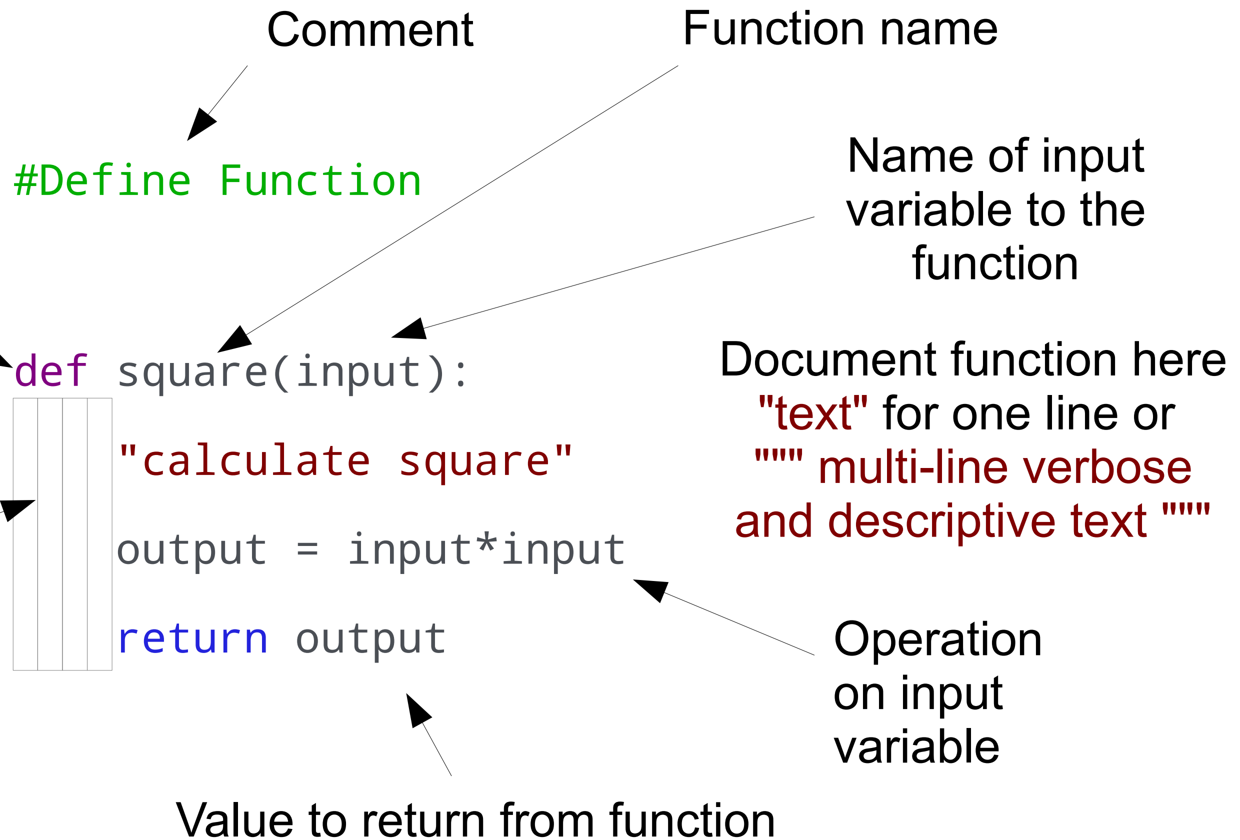
- Show how to use them in other constructs including conditionals (if statements) iterators (for loops) and functions (def name)
- Introduce external libraries numpy and matplotlib for scientific computing

Functions

```
#Define a variable
a = 5.0
```

Tell Python you are defining a function

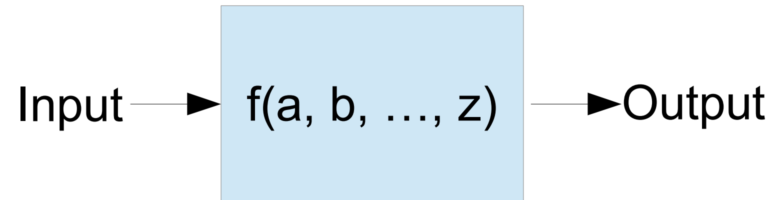
Level of indent determines what is inside the function definition. Variables defined (scope) exists only inside function. Ideally 4 spaces and avoid tabs. See PEP 8




```
#We call the function like this
square(a) Out: 25.0
```

Examples of Functions

- take some inputs
- perform some operation
- return outputs




```
def divide(a, b):
    output = a/b
    return output
```

```
def do_nothing(a, b):
    a+b
```

```
def get_27():
    return 27

#Call using
get_27()
```

```
def redundant(a, b):
    return b
```

```
def line(m, x, c=3):
    y = m*x + c
    return y
```

Optional variable. Given a value if not specified

```
def quadratic(a, b, c):
    "Solve: y = ax² + bx + c"
    D = b**2 + 4*a*c
    sol1 = (-b + D**0.5)/(2*a)
    sol2 = (-b - D**0.5)/(2*a)
    return sol1, sol2
```

Conditionals

- Allow logical tests

```
#Example of an if statement
if a > b:
    print(a)
else:
    print(a, b)


if type(a) is int:
    a = a + b
else:
    print("Error – a is type ", type(a))
```

Logical test to determine which branch of the code is run

```
if a < b:
    out = a
elif a == b:
    c = a * b
    out = c
else:
    out = b
```

Indent determine scope
4 spaces here

Strings

- String manipulations

```
s = "some string"
t = s + " with more"    Out: "some string with more"
s*3   Out: "some stringsome stringsome string"
s[3]                           Out: e
s[0:4]                         Out: some
s.title()                      Out: 'Some String'
s.capitalize()                 Out: "Some string"
s.find("x")                    Out: -1    #Not found
s.find("o")                    Out: 1
t = s.replace("some", "a")     Out: t="a string"
```

- In ipython, use tab to check what functions (methods) are avaliable

Reading Files (as Strings)

- Use with statement to ensure file is closed

```
#Get data from file
fdir = "C:/dir/" + "path/to/file/"
with open(fdir + './log.txt') as f:
    filestr = f.read()
#File is automatically closed on leaving 'with' scope
```

Reading the whole file is usually efficient but for large files may need to work through line by line:

```
f.readline()     #Reads to newline "\n" and increment file pointer
f.seek(0)        #Return to the start of the file
```

- Be careful of difference in functions: readline and readline**s**
- In ipython, use tab to check what functions (methods) are avaliable

Lists and iterators

- We can make lists of any type

```
m = ["another string", 3, 3.141592653589793, [5,6]]
print(m[0], m[3][0])      #Note indexing starts from zero
```

- Iterators – loop through the contents of a list

```
m = ["another string", 3, 3.141592653589793, [5,6]]
for item in m:
    print(type(item), " with value ", item)
```

- To add one to every element we could use

```
l = [1,2,3,4]
for i in range(len(l)):
    l[i] = l[i] + 1
```

Note: will not work:

```
for i in l:
    i = i + 1
```

List comprehension

l = [i+1 for i in l]

Dictionaries

- Dictonaries for more complex data storage

item

```
d = {"strings" : ["red", "blue"],
    "integers": 6,
    "floats": [5.0, 7.5]}
```

key   Value

```
e.items()

e.keys()
e.values()
```

- Access elements using strings

```
d["strings"]    out: ["red", "blue"]
```

- Elements can also be accessed using key iterators

```
for key in d:
    print(key, d[key])
```

Importing Numerical and Plotting Libraries

- Numpy – The basis for all other numerical packages to allow arrays instead of lists (implemented in c so more efficient)

```
import numpy as np
x = np.array([1,2,3]])
x = x + 1 # out np.array([2,3,4])
```

Import module numpy and name np
Similar to:
- c++ #include
- Fortran use
- R source()
- java import (I think...)
- MATLAB adding code to path

- matplotlib – similar plotting functionality to MATLAB

```
import matplotlib.pyplot as plt
plt.plot(x)
plt.show()    #Or to create a image file plt.savefig("out.png")
```

Use tab in ipython to see what code is available (or look online)

Classes in Python

- A person can train in a particular area and gain specialist skills

```
class Person():

    def __init__(self, name, age):

        self.name = name

        self.age = age

    def say_name(self):

        print("Hello, I'm "

              + self.name)
```

```
class Scientist(Person):
    def do_science(self):
        print(self.name +
              'is researching')
```

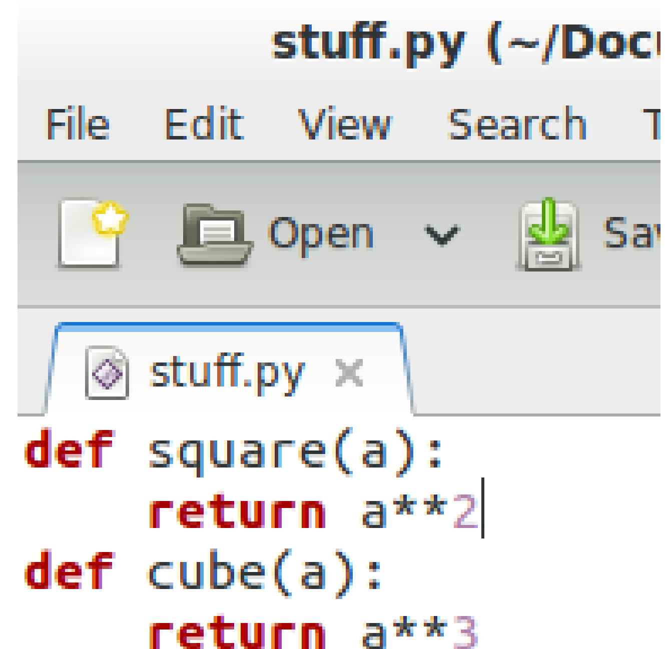
```
class Artist(Person):
    def do_art(self):
        print(self.name +
              'is painting')
```

```
bob = Artist('Bob Jones', 24)

jane = Scientist('Jane Bones', 32)

bob.do_art()

jane.do_science()
```

What is a Module?

- Simply copy code to a new file, for example `stuff.py`. Any script or Python session running in the same folder can import this,

```
import stuff

stuff.square(4.0)

stuff.cube(4.0)
```



- Module code should be functions and classes ONLY. Scripts to test/run can be included using the following:

```
if __name__ == "__main__":

    print(square(2.0), cube(2.0))
```

Tutors

- Chris Knight
- Isaac Sugden
- Edward Smith

- Ask the person next to you – there is a wide range of programming experience in this room and things are only obvious if you've done them before!

Hands-On Session 1

1) Functions – write a function to square inputs a and b and return their sum
2) Strings – Combined strings "hello" and " world", convert to capitals and print
3) Files – Open a plain text file (created with e.g. notepad) and print in Python
4) Lists – Create a list with 1,2 and 3, add an extra entry 4 and iterate through the list and print the contents
5) Dictionary – Create a shape_sides dictionary d with keys "triangle", "square" and "pentagon" and values 3, 4 and 5 respectivly. Iterate and print all items
6) Numpy arrays – Import the numpy module, create an numpy arrays of values from 1 to 5 and add one to each entry.
7) Create a module containing a function which adds two numbers a and b, returning thier sum. import into a script and print output
8) Classes – Create a class called number which takes an input x in its constructor and stores it (self.x = x). Add a method to square the (self.x) value and return

Introduction to Numpy and Plotting

Key Concepts – Arrays of data

- Python lists seem similar to arrays.They are not!

```
import numpy as np
m = [1,2,3,4,5,6]
x = np.array(m)
#Add one to a NumPy array increments elementwise
x = x + 1   # np.array([2, 3, 4, 5, 6, 7])
#But adding one to a list will cause a TypeError
m = m + 1
#But, conversion to numpy array if we mix types
x = x + m   #np.array([2, 4, 6, 8, 10, 12])
```

Methods for Numpy arrays

- Numpy arrays similar to MATLAB, Fortran, C++ std::array, R & (Java?)

```
import numpy as np
x = np.array([1,2,3])
```

- Numpy arrays have methods for statistical operations

```
x.mean()              (Note np.mean(x) equivalent)
x.std()              (Also np.std(x))
```

- While Numpy itself has a range of functions

```
np.median(x)     Out: 5.0  (But x.median doesn't work!!)
np.gradient(x)   Out: Numerical diff x_{i+1} - x_i (No x.gradient either)
```

- As with other objects, it pays to type ”x.” or ”np.” and use tab to see what is available, e.g.

```
newx = x.copy()    #Creates a copy of the array
```

Importing Numerical and Plotting Libraries

- matplotlib – similar plotting functionality to MATLAB

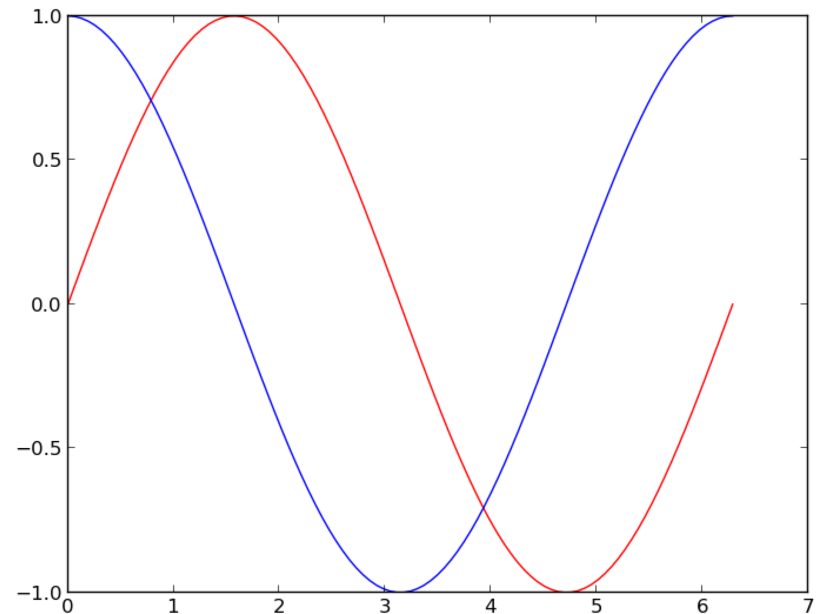
```
import matplotlib.pyplot as plt
x = np.array([0,1,1,2,3,5,8,13])
plt.plot(x)
plt.show()
#Or plt.savefig("out.png")
```

We need the pyplot submodule of matplotlib for most things. Dot uses plot/show from matplotlib.pyplot

Use tab in ipython to see what is available (or look online)

An Example plot

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


x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
z = np.cos(x)
plt.plot(x, y)
plt.plot(x, z)
plt.show()  #Or plt.savefig("out.png")
```



An Example plotting a histogram

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


#10,000 Uniform random numbers
x = np.random.random(10000)
#10,000 Normally distributed random numbers
y = np.random.randn(10000)
#Plot both on a histogram with 50 bins
plt.hist(y, 50)
plt.hist(x, 50)
plt.show()   #Or plt.savefig("out.png")
```

Lists vs Numpy Arrays

- Python Lists of lists seem similar to matrices.They are not!

```
m = [[1,2,3],[4,5,6],[7,8,9]]
m[0][1]    Out: 2
m[1][2]    Out: 6
```

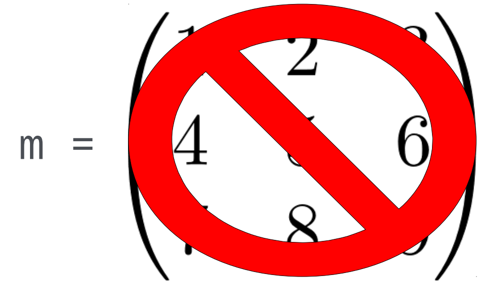
m = $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

Lists vs Numpy Arrays

- Python Lists of lists seem similar to matrices. They are not!

```
m = [[1,2,3],[4,5,6],[7,8,9]]
m[0][1]     Out: 2
m[1][2]     Out: 6
```

m =



Lists vs Numpy Arrays

- Python Lists of lists seem similar to matrices.They are not!

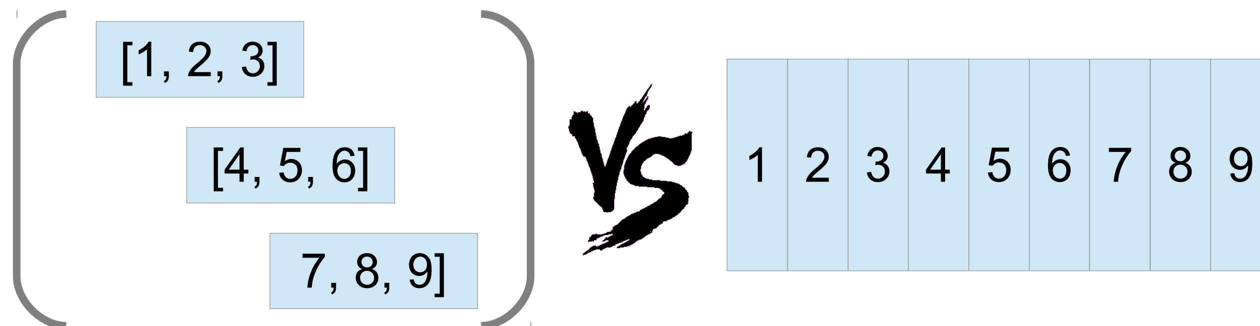
```
m = [[1,2,3],[4,5,6],[7,8,9]]
m[0][1]    Out: 2
m[1][2]    Out: 6
```

$$m = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- For numerics, use Numpy arrays which are contigous memory implemented in c (more efficient) and work like matrices

```
import numpy as np
x = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

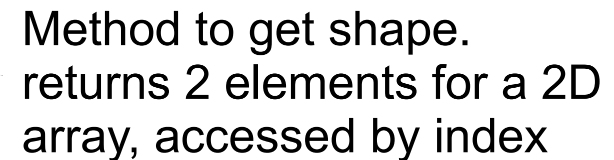
$$x = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$



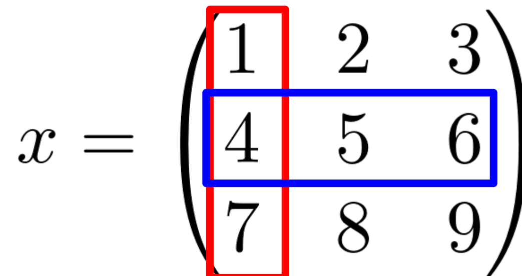
Array slicing

- Numpy arrays similar to MATLAB, Fortran, C++ std::array, R & (Java?)

```
import numpy as np
x = np.array([[1,2,3],[4,5,6],[7,8,9]])
for i in range(x.shape[0]):
    for j in range(x.shape[1]):
        print(i, j, x[i,j])
```




```
print(x[:,0])   #Out: Array([1, 4, 7])
print(x[1,:])   #Out: Array([4, 5, 6])
```



$$x = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Using Numpy arrays as Matrices

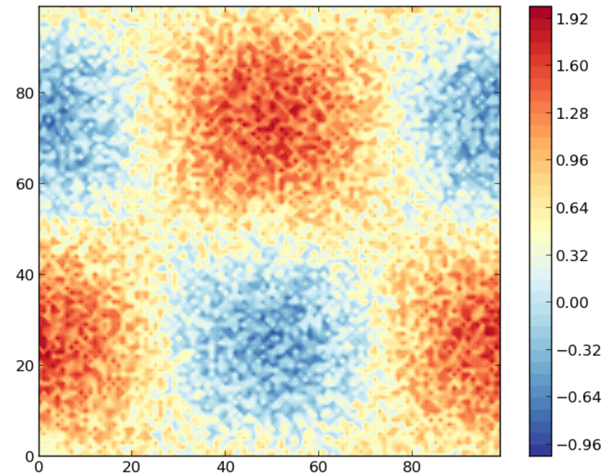
- Numpy has a number of array operations. As it is written in c, it is faster to perform operations with numpy instead of loops

```
import numpy as np
x = np.array([[1,2,3],[4,5,6],[7,8,9]])
y = x * 2       #Array operations
x.T             #Transpose array
x * y           #Elementwise (eqiv to MATLAB x .* y)
np.dot(x,y)     #Matrix multiply
# Invert matrix using linera algebra submodule of numpy
invy = np.linalg.inv(y)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- Numpy also has a np.matrix type which are for a special 2D version of Numpy arrays and provides options for linear algebra

An Example plotting a 2D Array (matrix)




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

N = 100
x = np.linspace(0,2*np.pi,N)
y = np.sin(x); z = np.cos(x)
#Create 2D field from outer product of previous 1D functions
noise = np.random.random(N**2)
u = np.outer(y,z) + noise.reshape(N,N)
plt.contourf(u, 40, cmap=plt.cm.RdYlBu_r)
plt.colorbar()
plt.show()  #Or plt.savefig("out.png")
```

Reshape an N**2 1D array into N by N 2D array

Creates a 2D array from two 1D arrays

Don't use Jet colormap

Hands on session 2

- Introduction
  1) Create a numpy array x=np.array([1,4,7]). Get the mean and standard deviation. Add one to each value in the array and get the new mean and standard deviation.
  2) Create array `x = np.array([[1,2,3],[4,5,6],[7,8,9]])`, use array slicing to get array([1, 2, 3]) and array([2, 5, 8]) and add them together.
  3) Setup a 3 by 3 identity matrix "I" (ones on the diagonal, zeros off diagonal). Create a 3 by 3 array of random numbers r. Check np.dot(I,r) is as expected
  4) Plot a tanh function in the range -2 p to 2 pi using linspace and matplotlib plot.
  5) Create a 1D array of 10,000 normally distributed random numbers t. Plot as a time history and zoom in to see the detail.
  6) Plot a histrogram of the array t from question 4) with 50 bins.
  7) Convert array t to a 2D array using field=t.reshape(100,100) and plot using contour.

Loading data from Files and Plotting

We Covered Reading Strings from Files Yesterday

- Use with statement to ensure file is closed

```
#Get data from file
with open('./file.csv') as f:
    filestr = f.read() #File is closed on leaving 'with' scope
```

file.csv

```
x,   y
1.0, 1.0
2.0, 4.0
3.0, 9.0
4.0, 16.0
5.0, 25.0
6.0, 36.0
```

We Covered Reading Strings from Files Yesterday

- Use with statement to ensure file is closed

```
#Get data from file
with open('./file.csv') as f:
    filestr = f.read() #File is closed on leaving 'with' scope


#Split into list using new line character ”\n”
lines=filestr.split("\n")
data = []
for line in lines[1:]:
    data.append(line.split(","))
```

What now? Lists of lists with strings...

plt.plot(data) will not work

file.csv

```
x,   y
1.0, 1.0
2.0, 4.0
3.0, 9.0
4.0, 16.0
5.0, 25.0
6.0, 36.0
```

An Plot Example Using Data from a csv File

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


#Read data from comma seperated variable file
data = np.genfromtxt("./file.csv", delimiter=',')


#Store columns as new variables x and y
x = data[:,0]
y = data[:,1]
plt.plot(x, y, "-or")
plt.show()
```

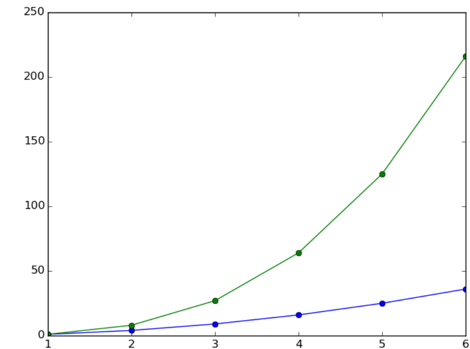
MATLAB syntax for plot line (-), point (o) in red (r)

file.csv

```
x,   y
1.0, 1.0
2.0, 4.0
3.0, 9.0
4.0, 16.0
5.0, 25.0
6.0, 36.0
```

An Example using data from a csv file + function

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


def read_file(filename):
    data = np.genfromtxt(filename, delimiter=',')
    x = data[:,0]; y = data[:,1]
    return x, y


for filename in ["sqr.csv", "cube.csv"]:
    x, y = read_file(filename)
    plt.plot(x, y, "-o")
plt.show()
```



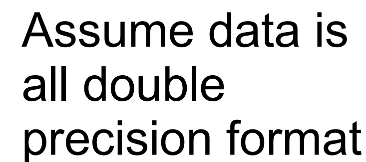
sqr.csv
```
x,   y
1.0, 1.0
2.0, 4.0
3.0, 9.0
4.0, 16.0
5.0, 25.0
6.0, 36.0
```

cube.csv
```
x,   y
1.0, 1.0
2.0, 8.0
3.0, 27.0
4.0, 64.0
5.0, 125.0
6.0, 216.0
```

Reading Data from a Binary File

- Reading Binary Format data

```
# Numpy helper function to read binary

f = "./binary/filename00001"

data = np.fromfile(open(f ,'rb'), dtype='d')
```



Reading files in other popular formats HDF5 or vtk

- Reading open-source HDF5 format (large binary data, self documenting) using python package h5py

```
import h5py
f = h5py.File(fpath,'r')
data = f[u'data'].items()[0][1]
```

- Another common format is vtk, open-source for 3D graphics visualization but I've had limited success reading: packages like vtk, pyvtk, mayavi/TVTK,

```
import vtk
reader = vtk.vtkUnstructuredGridReader()
reader.SetFileName(filename)
reader.ReadAllVectorsOn()
reader.ReadAllScalarsOn()
reader.Update()
```

Fasta data

“>” denotes records

```
>first sequence record
TACGAGAATAATTTCTCATCATCCAGCTTTAACACAAAATTCGCA

 >second sequence record
CAGTTTTCGTTAAGAGAACTTAACATTTTCTTATGACGTAAATGA
AGTTTATATATAAATTTCCTTTTTATTGGA

 >third sequence record
GAACTTAACATTTTCTTATGACGTAAATGAAGTTTATATATAAATTTCCTTTTTATTGGA
TAATATGCCTATGCCGCATAATTTTTATATCTTTCTCCTAACAAAACATTCGCTTGTAAA
```

Records have variable line length and sometimes newline in the record

Empty line between records

Fasta data

```
#Read data into a string
with open('fasta_file') as f:
   strs = f.read()
#Split into records assuming empty line between
records = strs.split("\n\n")
#Loop through records to get Dictionary, taking first
line of record as key and second line as value
d = {}
for r in records:
    indx = r.find("\n")
    value=r[indx+1:].replace("\n","")
    key = r[:indx].replace(">","")
    d[key] = value

# Use BioPython
from Bio import SeqIO
SeqIO.parse('fasta_file'
            , 'fasta')
```

```
>first sequence record
TACGAGAATAATTTCTCATCATCCAGCTTTAACACAAAATTCGCA

>second sequence record
CAGTTTTCGTTAAGAGAACTTAACATTTTCTTATGACGTAAATGA
AGTTTATATATAAATTTCCTTTTTATTGGA

>third sequence record
GAACTTAACATTTTCTTATGACGTAAATGAAGTTTATATATAAATTTCCTTTTTATTGGA
TAATATGCCTATGCCGCATAATTTTTATATCTTTCTCCTAACAAAACATTCGCTTGTAAA
```

An Example using data from a spreadsheet

```
import numpy as np


#Save data from spreadsheet into comma seperate file

data = np.genfromtxt("./sample_spreadsheet.csv", delimiter=',')
```

```
data = array([[ nan,   nan,   nan],
              [ nan,   27.,   78.],
              [ nan,   41.,   95.],
              [ nan,   22.,   55.],
              [ nan,   50.,  104.],
              [ nan,   45.,   82.],
              [ nan,   37.,  140.],
              [ nan,   84.,   50.]])
```

	A	B	C
1	Name	Age	Weight
2	Joe Bloggs	27	78
3	John Dow	41	95
4	Jane Doe	22	55
5	Gary Jones	50	104
6	Michael Hunt	45	82
7	James Brown	37	140
8	Jessica Green	84	50

An Example using data from a spreadsheet

```
import numpy as np
import matplotlib.pyplot as plt
#Save data from spreadsheet into comma seperate file
data = np.genfromtxt("./sample_spreadsheet.csv", delimiter=',')
#Plot data using array slicing
plt.plot(data[:,1], data[:,2],'o')
plt.show()
```

```
data = array([[ nan,  nan,  nan],
              [ nan,  27.,  78.],
              [ nan,  41.,  95.],
              [ nan,  22.,  55.],
              [ nan,  50., 104.],
              [ nan,  45.,  82.],
              [ nan,  37., 140.],
              [ nan,  84.,  50.]])
```

	A	B (data[:,1])	C (data[:,2])
1	Name	Age	Weight
2	Joe Bloggs	27	78
3	John Dow	41	95
4	Jane Doe	22	55
5	Gary Jones	50	104
6	Michael Hunt	45	82
7	James Brown	37	140
8	Jessica Green	84	50

An Example using data from a spreadsheet

```
import numpy as np

#Save data from spreadsheet into comma seperate file

data = np.genfromtxt("./sample_spreadsheet.csv", delimiter=',',
                     names=True)

data =
array([(nan, 27.0, 78.0),
       (nan, 41.0, 95.0),
       (nan, 22.0, 55.0),
       (nan, 50.0, 104.0),
       (nan, 45.0, 82.0),
       (nan, 37.0, 140.0),
       (nan, 84.0, 50.0)],
dtype=[('Name', '<f8'),
       ('Age', '<f8'),
       ('Weight', '<f8')])
```

	A	B	C
1	Name	Age	Weight
2	Joe Bloggs	27	78
3	John Dow	41	95
4	Jane Doe	22	55
5	Gary Jones	50	104
6	Michael Hunt	45	82
7	James Brown	37	140
8	Jessica Green	84	50

We can access with data['Age'] like a dictionary

An Example using data from a spreadsheet

```
import numpy as np
import matplotlib.pyplot as plt
#Save data from spreadsheet into comma seperate file
data = np.genfromtxt("./sample_spreadsheet.csv", delimiter=',',
                     names=True)
#Plot data using name keywords
plt.plot(data['Age'], data['Weight'], 'o')
plt.show()
```

	A	B	C
1	Name	Age	Weight
2	Joe Bloggs	27	78
3	John Dow	41	95
4	Jane Doe	22	55
5	Gary Jones	50	104
6	Michael Hunt	45	82
7	James Brown	37	140
8	Jessica Green	84	50

An Example using data from a spreadsheet

```
import numpy as np


#Save data from spreadsheet into comma seperate file

data = np.genfromtxt("./sample_spreadsheet.csv", delimiter=',',
                     names=True, dtype=object)
```

```
data =
array([('Joe Bloggs', '27',
'78'), ('John Dow', '41', '95'),
('Jane Doe', '22', '55'), ('Gary
Jones', '50', '104'),('Michael
Hunt', '45', '82'), ('James
Brown', '37', '140'),('Jessica
Green', '84', '50')],
dtype=[('Name', 'O'),
       ('Age', 'O'),
       ('Weight', 'O')])
```

	A	B	C
1	Name	Age	Weight
2	Joe Bloggs	27	78
3	John Dow	41	95
4	Jane Doe	22	55
5	Gary Jones	50	104
6	Michael Hunt	45	82
7	James Brown	37	140
8	Jessica Green	84	50

An Example using data from a spreadsheet

```
import numpy as np

import matplotlib.pyplot as plt

#Save data from spreadsheet into comma seperate file

data = np.genfromtxt("./sample_spreadsheet.csv", delimiter=',',

                     names=True, dtype=object)

#Numpy arrays must all be the same type and currently an array of
objects which we need to convert to allow plotting

plt.plot(data['Age'].astype("float"),

         data['Weight'].astype("float"), 'o')

plt.show()
```

Pandas example with a spreadsheet

```
import matplotlib.pyplot as plt

import pandas

data = pandas.read_excel("./sample_spreadsheet.xlsx")
```

```
data =         Name  Age  Weight
0        Joe Bloggs   27      78
1          John Dow   41      95
2          Jane Doe   22      55
3        Gary Jones   50     104
4      Michael Hunt   45      82
5       James Brown   37     140
6     Jessica Green   84      50
```

	A	B	C
1	Name	Age	Weight
2	Joe Bloggs	27	78
3	John Dow	41	95
4	Jane Doe	22	55
5	Gary Jones	50	104
6	Michael Hunt	45	82
7	James Brown	37	140
8	Jessica Green	84	50

```
#Some example operations

data.boxplot(); plt.show()  #Can call inbuilt plots

data.corr()    # Look at correlations
```

	Age	Weight
Age	1.000000	-0.246889
Weight	-0.246889	1.000000

Save Data in Python's own format using pickle

- Import and save data from Python in any python format

```
import pickle
a = 4.
s = "test"
l = [2,3,4]
d = {"stuff":2}
pickle.dump([a, s, l, d],open('./out.p','w'))
```

- Then in a different script or session of Python we can load any of these types in the right format

```
import pickle
a, s, l, d = pickle.load(open('./out.p','r'))
```

Hands-On Session 3

Introductory Questions

1) Setup a 2D matrix z = np.array[[1,2,3,4],[1,4,9,16]]. Use array slicing (or a loop) to get two arrays a=[1,2,3,4] and b=[1,4,9,16] and plot them against each other.
2) Create a csv file (using excel or a text editor) to give two columns containing the data from part 1), read into Python and plot one column against the other
3) Open a spreadsheet (e.g. the sample in the examples folder) using either pandas or convert to csv and use genfromtxt and plot (e.g. age against weight).
4) Use Pickle to dump list [4,6,7] and string ”hello”. Load in a new script/session

Advanced Question

5) Write a function to read a csv file as a string and convert to numbers stored in a numpy array. What options do you need to make it more general (e.g. to skip header lines). Use on question 3) and 4) above.
6) Add names option to take the title on the top column and create a dictionary of arrays

Using Python as glue: Filesystems, subprocess and ctype

Using os.system

- Simple external commands can be called with the os system

```
import os
os.system("echo 1 > file")
os.mkdir("new_folder")
```

- Changing directory can also be done with os

```
import os
cwd = os.getcwd()  #Save current directory
os.chdir("./new_folder")  #Got to new directory
os.system("echo 2 > newfile")
os.chdir(cwd)   #Go back to previous directory
```

Manipulating the File System

- Getting all files in a directory can be done with glob (returns a list)

```
import glob
# Get contents of directory "./" using wildcard *
files = glob.glob("./*")
# Then iterate through list of strings
for file in files:
    print(file)
```

- Copying, moving or deleting files can be done with shutil (cross platform)

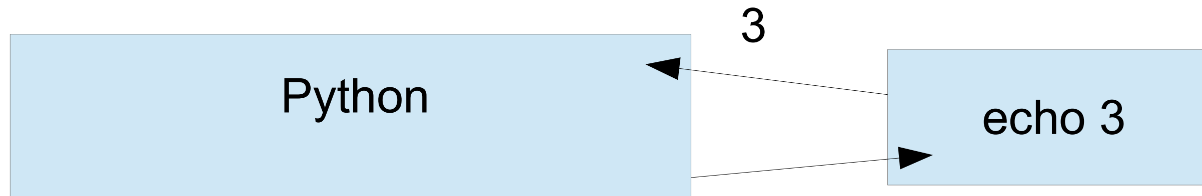
```
import shutil
shutil.copyfile("path/to/file", "new/path")
shutil.rmtree("folder/to/remove")
```

Running Jobs Using Subprocess

- Introduction to syntax to run an executable from within Python, in this example echo

```
import subprocess


# Call any executable
output = subprocess.check_output("echo 3", shell = True)
```

Passing commands separated by spaces as a list, e.g.
["echo", "3"]
would work without shell



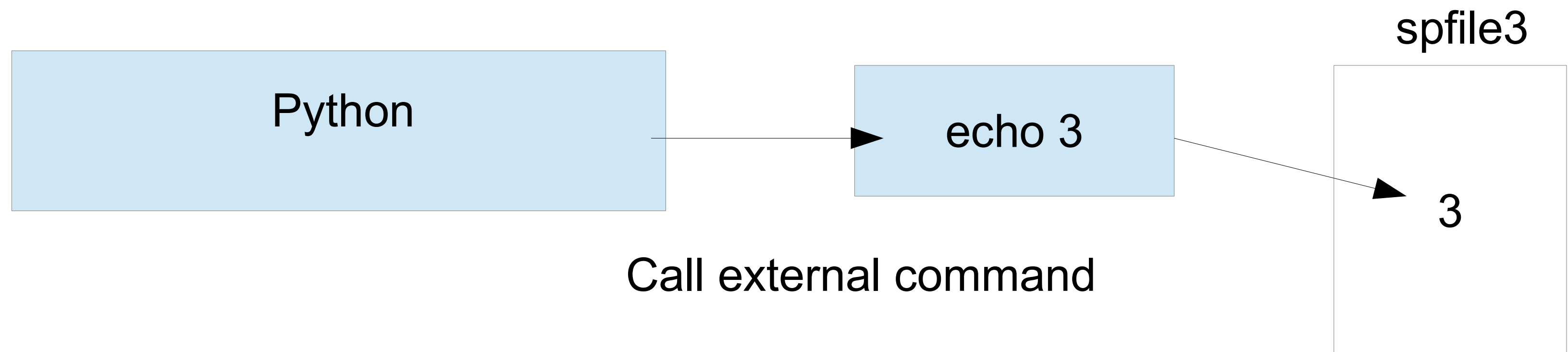
Call external command

Running Jobs Using Subprocess

- Introduction to syntax to run an executable from within Python, in this example echo

```
import subprocess

# Call any executable
sp = subprocess.Popen("echo 3 > spfile3", shell = True)
```

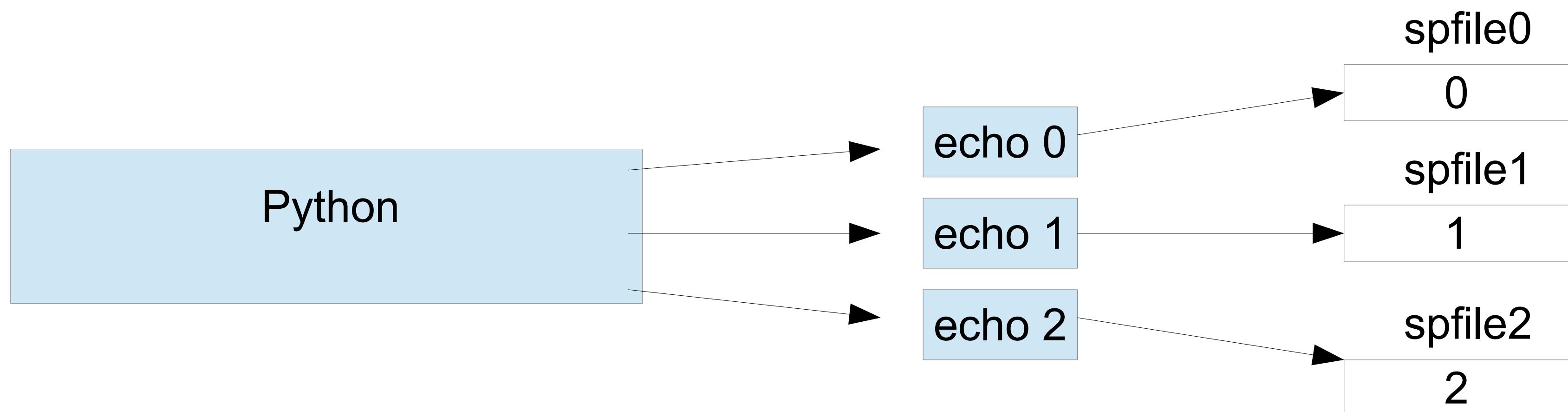
Needed to interpret shell command but security risk as any string command could be run



Running Jobs Using Subprocess

- Introduction to syntax to run an executable from within Python, in this example echo

```
import subprocess
for i in range(3):
    sp = subprocess.Popen("echo " + str(i) + " > spfile"
                          + str(i), shell = True)
```



Running C/C++ Using Subprocess

- Introduction to syntax to run an executable written in c++ and compiled

```
from subprocess import Popen, PIPE
# Call C++ executable ./a.out
sp = Popen(['./a.out'], shell=True,
           stdout=PIPE, stdin=PIPE)
# test value of 5
value = 5
# Pass to program by standard in
sp.stdin.write(str(value) + '\n')
sp.stdin.flush()
# Get result back from standard out
result = sp.stdout.readline().strip()
print(result)
```

```
//test.cpp code to add
//1 to input and print

#include <iostream>
using namespace std;

int add_one(int i)
{
    return i+1;
}

int main () {

    int i;
    cin >> i;
    i = add_one(i);
    cout << i << "\n";
}
```

Python

a.out

a.out compiled using:
g++ test.cpp

Combining Commands

We can automate a wide range of task with these techniques

- Move to a directory somewhere

```
fdir = "C:/dir/" + "path/to/runs/"
```

- Download a software file from a website and unzip using Python tarfile/gunzip libraries or os.system/subprocess

```
os.system("wget http://www.somefile.com") or os.system("git clone ...")
```

- Build software, use error handling
- Copy a snapshot of the src code and all input files to run directory
- Run a parameter sweep by tweaking input files with Python
- Summarise and plot results

```
try:
    subprocess.check_output("configure")
    subprocess.check_output("make")
except subprocess.CalledProcessError as e:
    print(e.output)
```

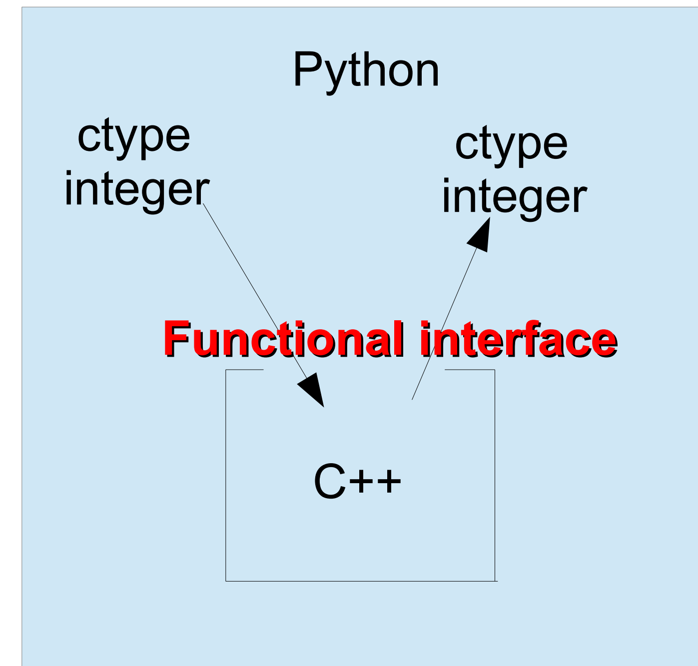
Discussion of Wrapping Code with ctypes

- Creates an interface for your C or C++ function

```
extern "C" int add_one(int i)
{
    return i+1;
}
```

- Python code to use this function is then

```
import numpy.ctypeslib as ctl
import ctypes

libname = 'testlib.so'
libdir = './'
lib=ctl.load_library(libname, libdir)

py_add_one = lib.add_one
py_add_one.argtypes = [ctypes.c_int]
value = 5
results = py_add_one(value)
print(results)
```



Compile to shared library with: g++ -shared -o testlib.so -fPIC test.cpp

Ctypes Example of fast loop

- We can use for a function which would be slow in Python (e.g. check prime numbers)

```
def Pyisprime(number):
    if (number < 1):
        return 0
    for i in range(2, int(number**0.5)):
        if number%i == 0:
            return 0
    return 1
```

- Same function in C with the same interface

```
extern "C" int isprime(unsigned int number) {
    if (number <= 1) return 0; // zero and one are not prime
    unsigned int i;
    for (i=2; i*i<=number; i++)
        if (number % i == 0) return 0;
    return 1;
}
```

Compile with: g++ -shared -o isprime.so -fPIC isprime.cpp

Ctypes Example of fast loop

- Function in C

```
extern "C" int isprime(unsigned int number) {
    if (number <= 1) return 0; // zero and one are not prime
    unsigned int i;
    for (i=2; i*i<=number; i++)
        if (number % i == 0) return 0;
    return 1;
}
```

Compile with: g++ -shared -o isprime.so -fPIC isprime.cpp

- Python code to wrape this function

```
import numpy.ctypeslib as ctl
import ctypes
lib=ctl.load_library("isprime.so", "./")
Cisprime = lib.isprime
Cisprime.argtypes = [ctypes.c_uint]
for i in range(100):
    if Cisprime(i) is 1:
        print(i, " is a prime number")
```

Ctypes Example of fast loop

- We can them compare the two functions over 10,000 loops

```
import timeit

#Test the C function
start_time = timeit.default_timer()
for i in range(10000):
    isprime(95725787+i)
print("C Times = ", timeit.default_timer() - start_time)

#Test the Python function
start_time = timeit.default_timer()
for i in range(10000):
    Pyisprime(95725787+i)
print("Python Time = ", timeit.default_timer() - start_time)
```

Threading and MPI

- Subprocess can run many jobs
  - Can be used to start as many independent jobs in parallel as you have processor cores
  - Independent processes with only shared access to disk – read/write data to files
  - Popen is non-blocking but you can use sp.wait()
- Python does not always use your multi-core processor efficiently – global interpreter lock (GIL)
- The multiprocessor library can be used to explicitly divide shared memory jobs (similar to OpenMP)
- Rewriting in C code can also better utilise resources, Numpy does this
- On distributed memory platforms, MPI can be used with Python through mpi4py

Hands-On Session 4

Introduction

1) Use glob to get files in the current directory, loop through list and print all files. Create a list of only the python scripts (i.e. files which end with .py).

2) Use os to create a folder and change directory to it. Use a loop to create filesname0 to filename10 (see hands on 3 yesterday) each file containing the number 0 to 10 respectivly (note os.system(”echo 5 > filename5”) creates a 5 in filename5

3) Read the contents of files filename0, …, filename10 either using Python open or subprocess ("cat filename0" in linux/mac, "type filename10" in windows)

4) Use subprocess instead of os.system in 2), read using 3) and check

Advanced

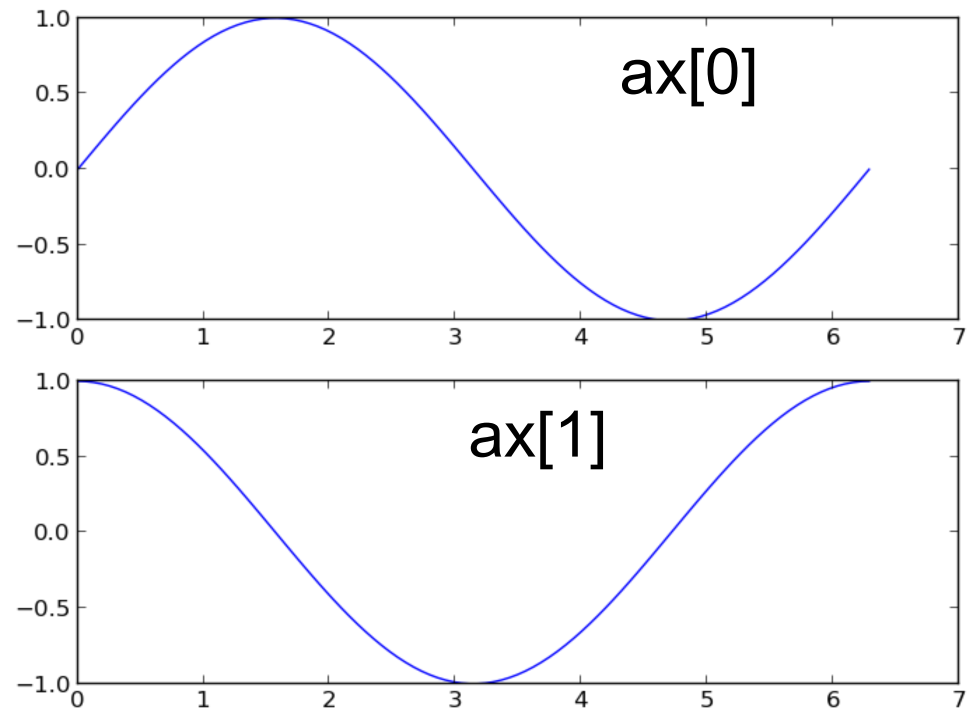
5) Compile a low level code (C, Fortran or other) and run using subprocess returning the output to Python

6) Write a simple c function which takes an float, subtracts 1.0 and returns. Write a ctypes wrapper in Python and call it. What do you notice about duck typing here?

More Advanced Plotting

A Plot of Two Axes

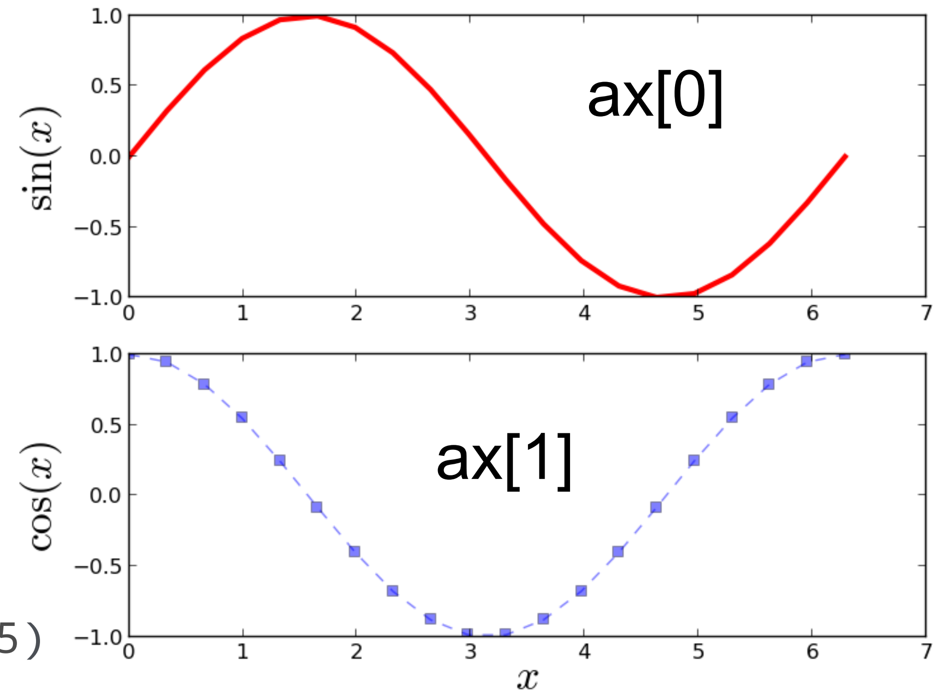
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
z = np.cos(x)
fig, ax = plt.subplots(2,1)
ax[0].plot(x, y)
ax[1].plot(x, z)
ax[1].set_xlabel("x axis", fontsize=24)
plt.show()
```



Two subplots, ax is a list of so called axis handles and we use the plot method of these handles.

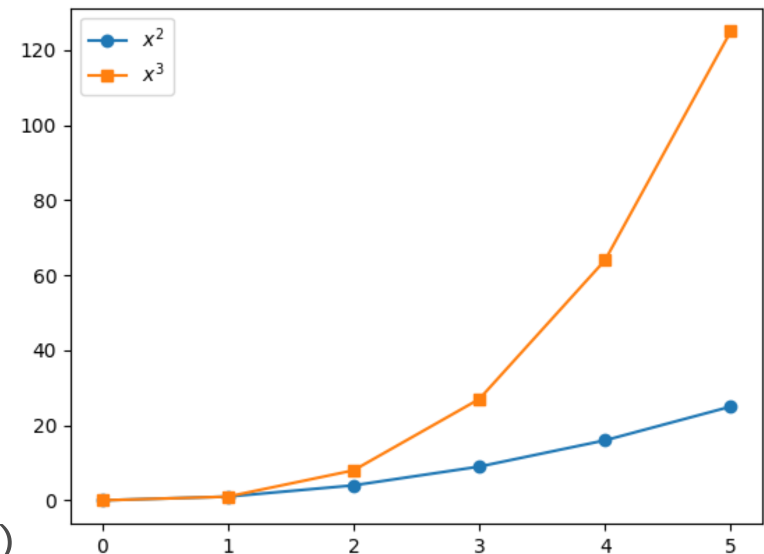
A Plot of Two Axes with Labels and Styles

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 20)
y = np.sin(x)
z = np.cos(x)
fig, ax = plt.subplots(2,1)
ax[0].plot(x, y, lw=3., c='r')
ax[1].plot(x, z, '--bs', alpha=0.5)
ax[1].set_xlabel("$x$", fontsize=24)
ax[0].set_ylabel("$\sin(x)$", fontsize=24)
ax[1].set_ylabel("$\cos(x)$", fontsize=24)
plt.show()
```



An Example with a legend

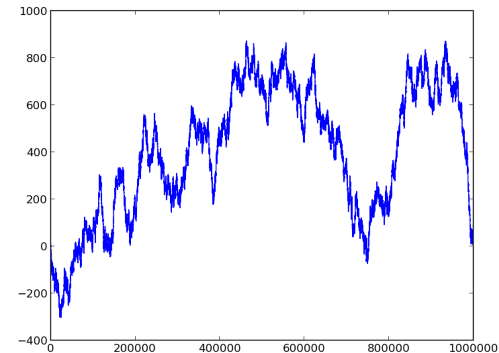
```
import numpy as np
import matplotlib.pyplot as plt


#Get six values as a numpy array
x = np.arange(6)
#Plot with latex syntax
plt.plot(x, x**2, "-o", label="$x^2$")
plt.plot(x, x**3, "-s", label="$x^3$")
plt.legend()
plt.show()
```

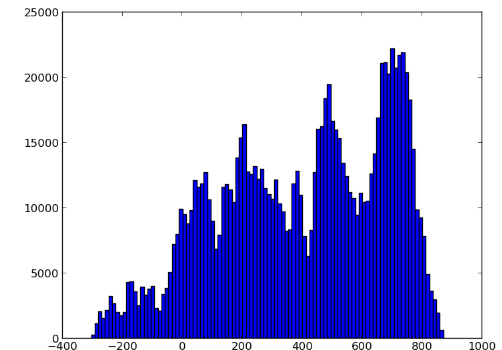


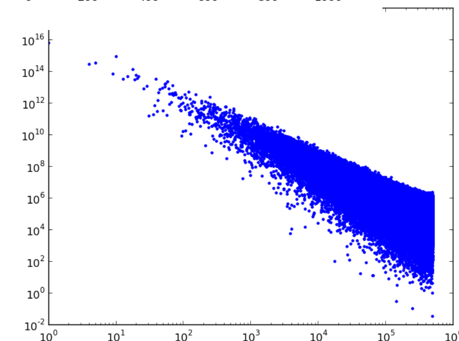
Imperial College London

An Example using time series

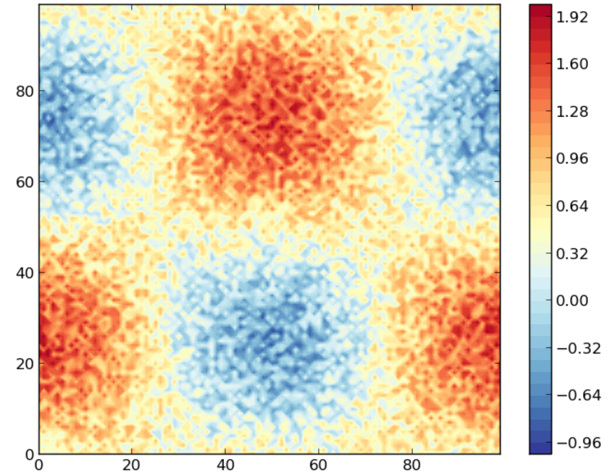
```
import numpy as np
import matplotlib.pyplot as plt


N = 1000000
signal = np.cumsum(np.random.randn(N))
plt.plot(signal); plt.show()
plt.hist(signal, 100); plt.show()
Fs = np.fft.fft(signal)**2
plt.plot(Fs.real[:N/2], ".")
plt.xscale("log"); plt.yscale("log")
plt.show()
```







An Example Plotting a 2D Field (matrix)

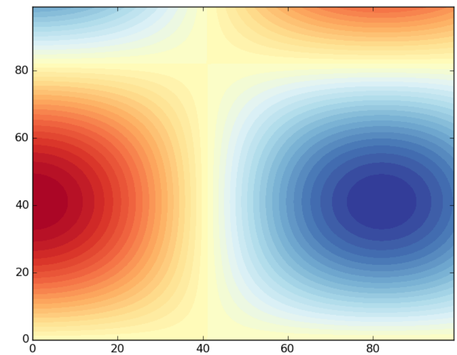


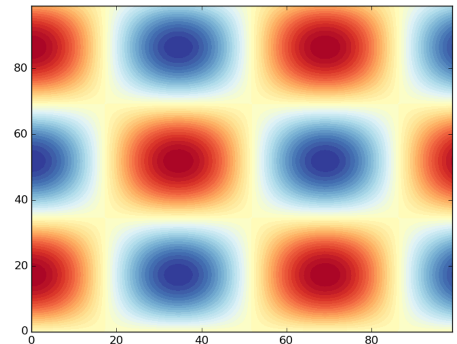

```
import numpy as np
import matplotlib.pyplot as plt

N = 100
x = np.linspace(0,2*np.pi,N)
y = np.sin(x); z = np.cos(x)
#Create 2D field from outer product of previous 1D functions
u = np.outer(y,z) + np.random.random([N,N])
plt.contourf(u, 40, cmap=plt.cm.RdYlBu_r)
plt.colorbar()
plt.show()
```

Don't use Jet colormap

An Example of Animation

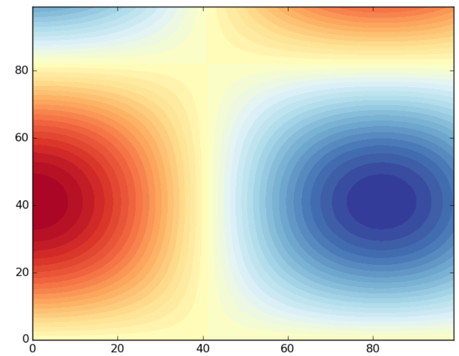
```
import numpy as np
import matplotlib.pyplot as plt
def get_field(a, N = 100):
    x = a*np.linspace(0,2*np.pi,N)
    y = np.sin(x); z = np.cos(x)
    return np.outer(y,z)
plt.ion(); plt.show()   #Interactive plot
for i in np.linspace(0., 5., 200):
    u = get_field(i)    #Call function with new
    plt.contourf(u, 40, cmap=plt.cm.RdYlBu_r)
    plt.pause(0.01)   #Pause to allow redraw
    plt.cla()   #Clear axis for next plot
```

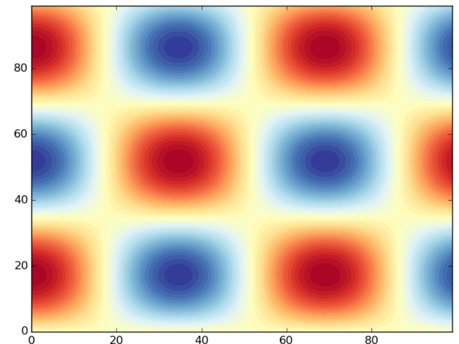




An Example of making a video

```
import numpy as np
import matplotlib.pyplot as plt
def get_field(a, N = 100):
    x = a*np.linspace(0,2*np.pi,N)
    y = np.sin(x); z = np.cos(x)
    return np.outer(y,z)
plt.ion(); plt.show()   #Interactive plot
for n, i in enumerate(np.linspace(0., 5., 200))
    u = get_field(i)    #Call function with new
    plt.contourf(u, 40, cmap=plt.cm.RdYlBu_r)
    plt.pause(0.01)   #Pause to allow redraw
    plt.savefig("filename{:05}".format(n), bbox_inches="tight")
    plt.cla()   #Clear axis for next plot
```

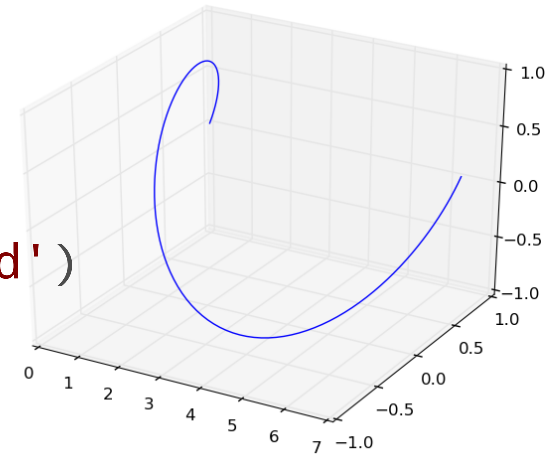




Three dimensional plots in matplotlib vs. mayavi

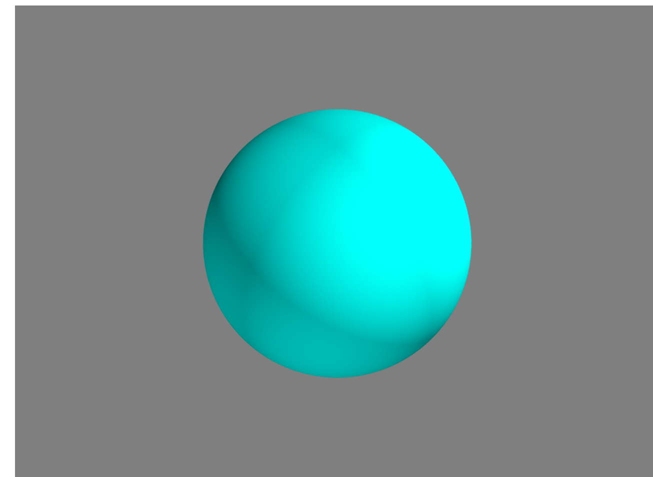
- Some 3D plotting in matplotlib (but limited)

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = np.linspace(0.,2*np.pi,100)
ax.plot(x, np.cos(x), np.sin(x))
plt.show()
```



- Generate isosurface data using mayavi (better 3D than matplotlib)

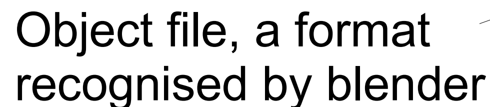
```
import numpy as np
import mayavi.mlab as mlab
x = np.linspace(-1., 1., 100)
y = x; z = y
[X,Y,Z] = np.meshgrid(x,y,z)
out1 = mlab.contour3d(X**2+Y**2+Z**2,
                      contours=[0.8])
mlab.show()
```



Three dimensional plots in mayavi

- Generate isosurface data using mayavi from 3D postproc reader

```
import mayavi.mlab as mlab

#3D DATA FIELDS LOADED HERE
# .......................
#3D DATA FIELDS LOADED HERE

for i in range(minrec,maxrec):
    field = pp.get_field(i)
    out1 = mlab.contour3d(field, contours=[0.3])
    mlab.savefig('./surface{:05d}'.format(i)+'.obj')
```



Blender python interface

- Use python plugin to import isosurface, set material and save render

```
import bpy, bmesh
#Blender file saved with correctly setup camera/light source
bpy.ops.wm.open_mainfile('./scene.blend')
for i in range(minrec,maxrec):
    #Load object file from mayavi
    file ='/surface{:05d}'.format(i)
    bpy.ops.import_scene.obj(file+'.obj')
    obj = bpy.context.selected_objects[:][0]
    #Set material and render
    mat = bpy.data.materials['shiny_tranparent']
    obj.data.materials[0] = mat
    bpy.data.scenes['Scene'].render.filepath =file+'.jpg'
    bpy.ops.render.render( write_still=True )
    #Delete last object ready to load next object
    bpy.ops.object.select_all(action='DESELECT')
    bpy.context.scene.objects.active = obj
    obj.select = True
    bpy.ops.object.delete()
```

Blender Videos

A GUI with a Slider

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

#Setup initial plot of sine function
x = np.linspace(0, 2*np.pi, 200)
l, = plt.plot(x, np.sin(x))

#Adjust figure to make room for slider
plt.subplots_adjust(bottom=0.15)
axslide = plt.axes([0.15, 0.05, 0.75, 0.03])
s = mw.Slider(axslide, 'A value', 0., 5.)

#Define function
def update(A):
    l.set_ydata(np.sin(A*x))
    plt.draw()

#Bind update function to change in slider
s.on_changed(update)
plt.show()
```

Adjust figure to make room for the slider and add a new axis axslide for the slider to go on

Define a function to change figure based on slider value. Here this updates the plot data and redraws the plot

Bind function update to slider change

Curve Fitting with Scipy

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

x = np.linspace(0, 4., 30)
y = x + (2.*(np.random.random(30)-.5))
plt.plot(x, y, 'ob')

def linear(x, m, c):
    "Define line function"
    return m*x + c

params, cov = curve_fit(linear, x, y)
yf = linear(x, params[0], params[1])
plt.plot(x, yf, 'r-')

plt.show()
```

Function from scipy. Takes function handle for the fit you want with x and y data. It returns fit parameters (here m and c) as a list with 2 elements and the covariance (for goodness of fits, etc)

We use params (m and c) with the linear function to plot the fit

Hands-On Session 5

1) Create x=np.linspace(0., 10.,1000) and plot x^2 and x^3 on axes ax[0] and ax[1] from plt.subplots(2,1). Change line colour, markers and size
2) Change the y axes on 1) to logarithmic and label the x and y axes
3) Create 2D data from 1D arrays y and z using x=np.outer(y,z) and plot using imshow, contouf or pcolormesh (try different 1D arrays)
4) Fit an appropriate line to

x = np.linspace(0, 2*np.pi, 100) and

y =np.sin(x) + (2.*(np.random.random(100)-.5))

<u>Advanced</u>

5) Create fig, ax = plt.subplots(1,1), switch interactive mode on and plot ax.plot(np.sin(A*x)) for A in np.linspace(-5,5,100) using plt.pause(0.1) to redraw and plt.cla() to clear the axis (NOTE WON'T WORK IN NOTEBOOK AND WILL NEED TO SAVE FILES IN PYTHONANYWHERE)
6) Run the slider example and adapt to plot sin(Ax^2) using function from number, num.square, with the value of A specified by the slider value.
7) Develop a slider example with both sine and cosine on the plot updated by slider. Adapt this to add a new slider for a second coefficient B for cos(Bx).

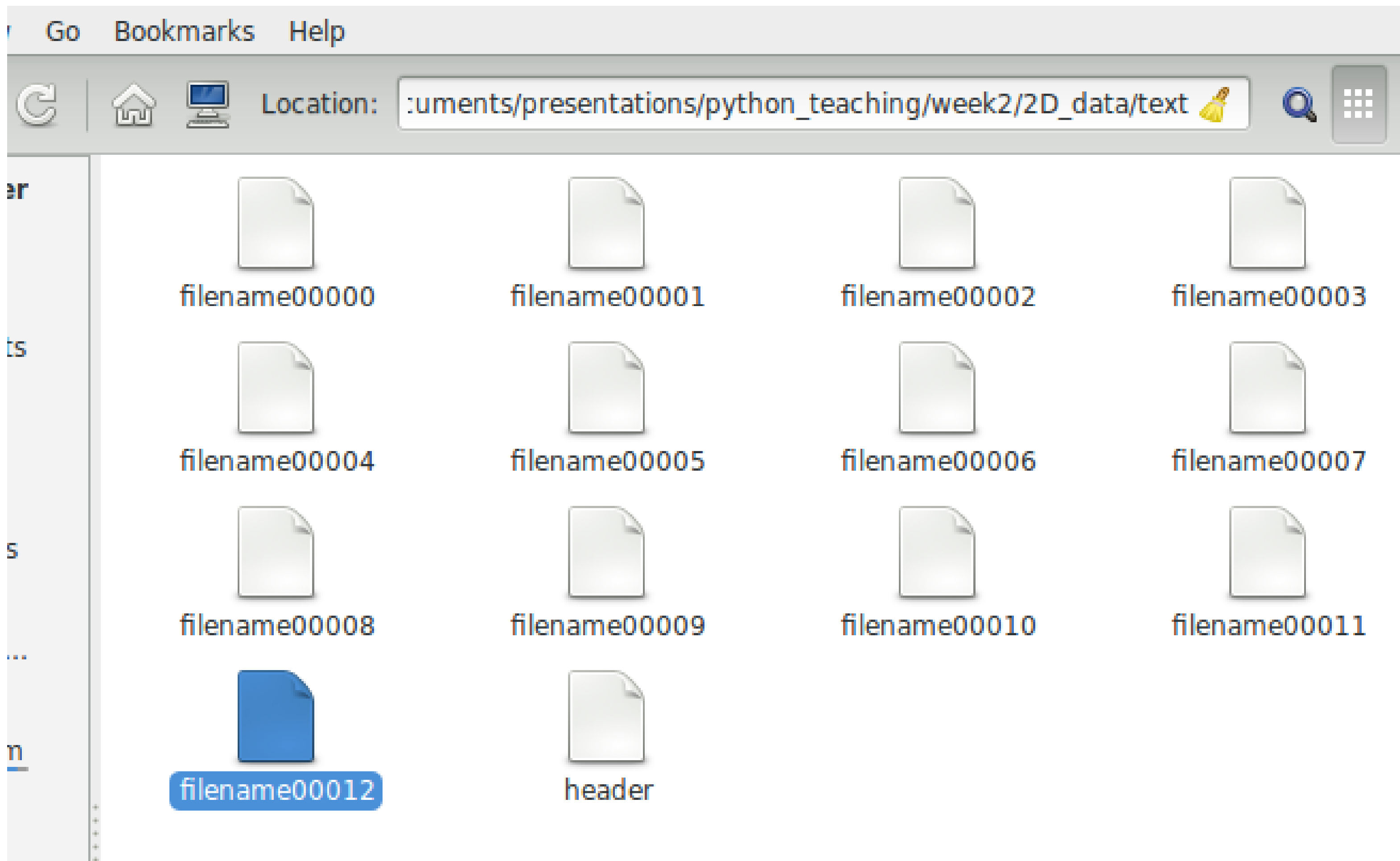
A Complete Post-Processing Example

A Typical Postprocessing Workflow

- Get data from some source: experiments, numerical simulation, surveys/studies, an internet database, etc.
- Import it into python as a single numpy array, a list of numpy arrays, a dictionary of values, etc.
- Play around with various plots and data analysis techniques.
- Take the most promising output and save the script which generates this exactly, add labels and format to publication quality.
- We can develop an automated process from data to figure with minimal user input. This is useful because
  - Easy to make changes when required by reviewers
  - Clearer mapping from data to output (opendata movement)
  - Create functions to break the analysis down and reduce errors
  - You can use the same scripts to analyse similar data

A Practical Example of Plotting

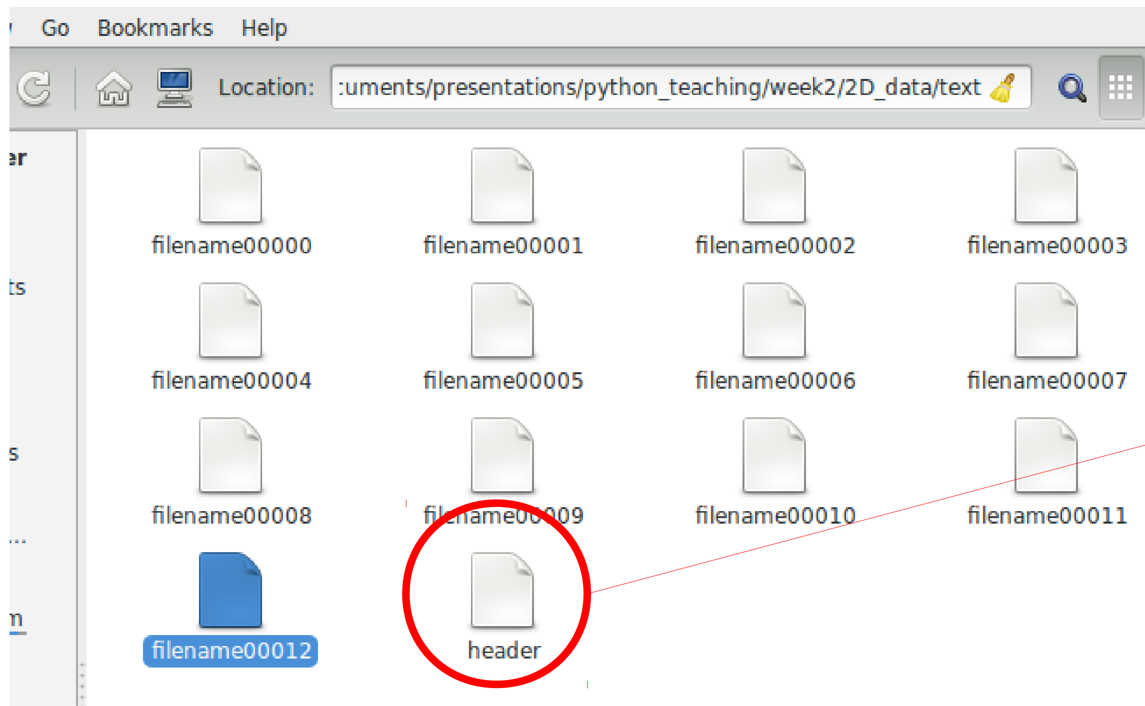
We have multiple 2D images stored in files with a header file



A Practical Example of Plotting

We have 2D data in multiple files with header (meta-data). We want

1) A function to read the header file and store parameters



header file

Nx 84
Nz 50
Lx 1560.41523408474
Lz 1069.90830902657
Nrecs 12

Reading from files

- Opening and finding keywords in file

```
#Find a keyword in file and read numbers to the right
with open('./header') as f:
    for l in f.readlines():
        if l.find("timestep") != -1:
            dt = float(l.strip('timestep'))
            break
```

header file

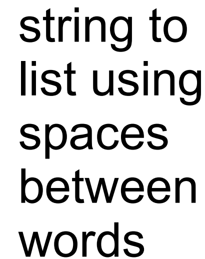
```
Nx 84
Nz 50
Lx 1560.41523408474
Lz 1069.90830902657
Nrecs 12
```

- But assumes we know keywords to look for in data

Reading into Dictionaries

- Dictonaries are ideal for reading meta-data

```
header = {}
f = open('./header')
for l in f.readlines():
    key, value = l.split()
    header[key] = float(value)
```



header file

Nx 84
Nz 50
Lx 1560.41523408474
Lz 1069.90830902657
Nrecs 12

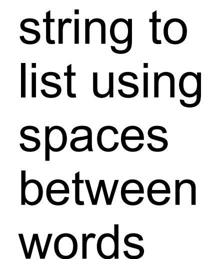
Iterate through values saved in header file and print

```
for key, value in header.items():
    print(key, value)
```

Reading into Dictionaries

Data is a mix of integers and floats, we can use error handling to get type

```
header = {}
f = open('./header')
for l in f.readlines():
    key, value = l.split()
    try:
        header[key] = int(value)
    except ValueError:
        header[key] = float(value)
```



header file

```
Nx 84
Nz 50
Lx 1560.41523408474
Lz 1069.90830902657
Nrecs 12
```

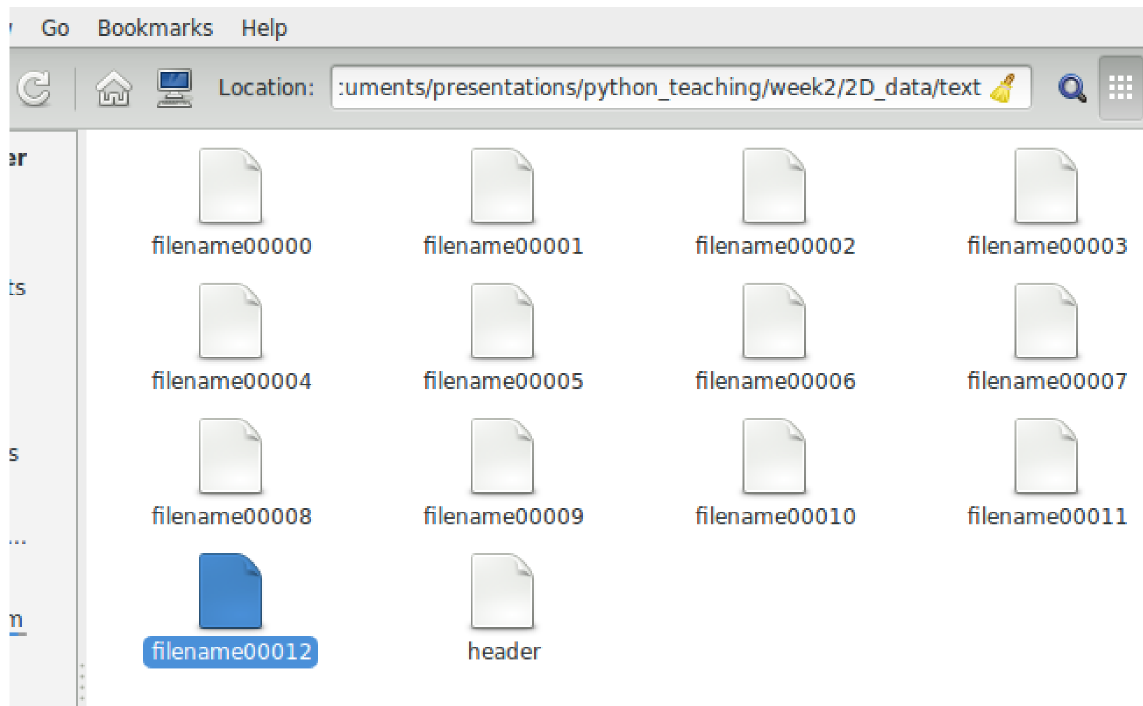
A Practical Example of Plotting

We have 2D data in multiple files with header (meta-data). We want

1) A function to read the header file and store parameters

2) A function to get the list of data files in the folder

```
for i in range(10):
    print("filename0000"
          + str(i))
```




```
"filename00000"
"filename00001"
"filename00002"
"filename00003"
"filename00004"
.........
```

Get All Files in Folder

- Write a loop to print 10 strings with names: "filename0", "filename1", … "filename9" (note str(i) converts an int to a string)

```
for i in range(10):
    print("filename0000" + str(i))
```

- More useful is the format method, with prepended zeros, so files are in displayed in order in folder (and read in order):

```
for i in range(13):
    print("filename{:05}".format(i))
```

- Get contents of all folder with same name

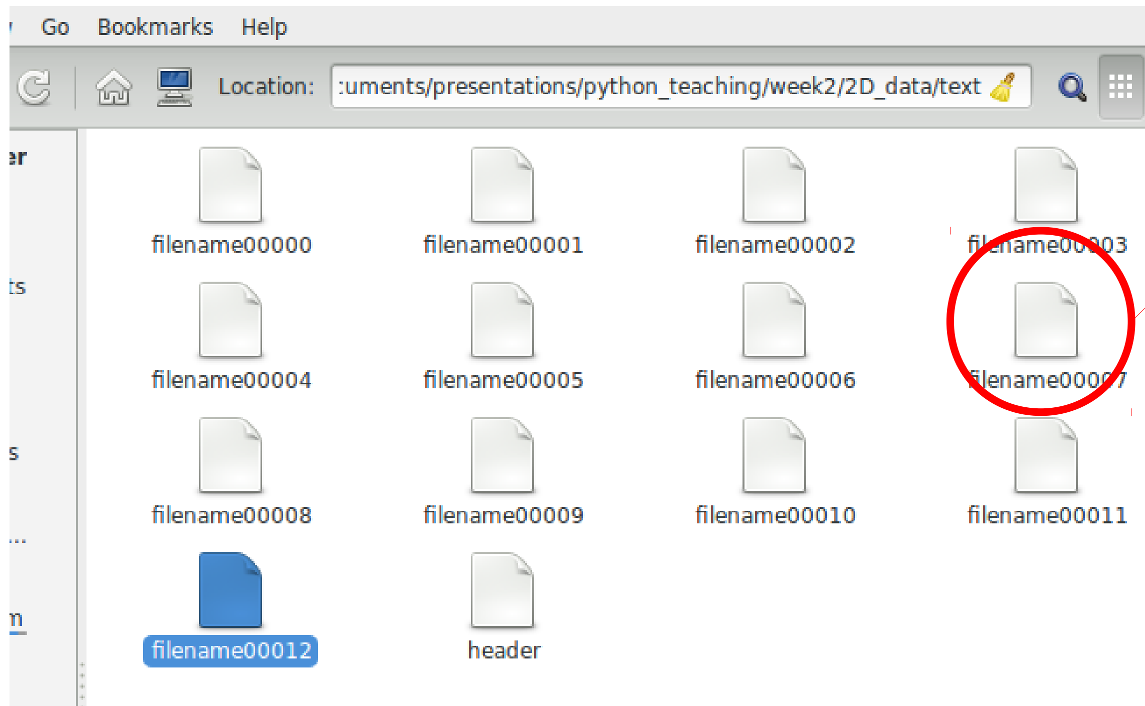
```
import glob
for i in glob.glob("filename*"):
    print(i)
```

A Practical Example of Plotting

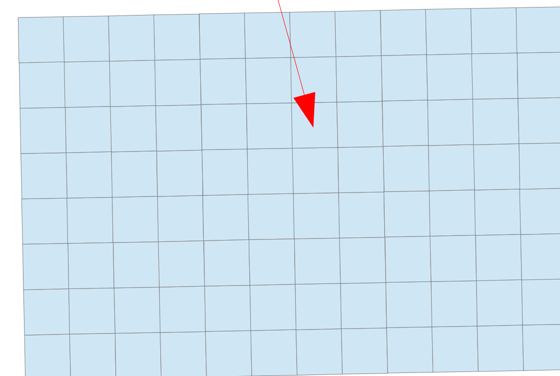
We have 2D data in multiple files with header (meta-data). We want

1) A function to read the header file and store parameters
2) A function to get the list of data files in the folder
3) A function to read data

The data in each file is 4200 floats describing a 84 by 50 2D field

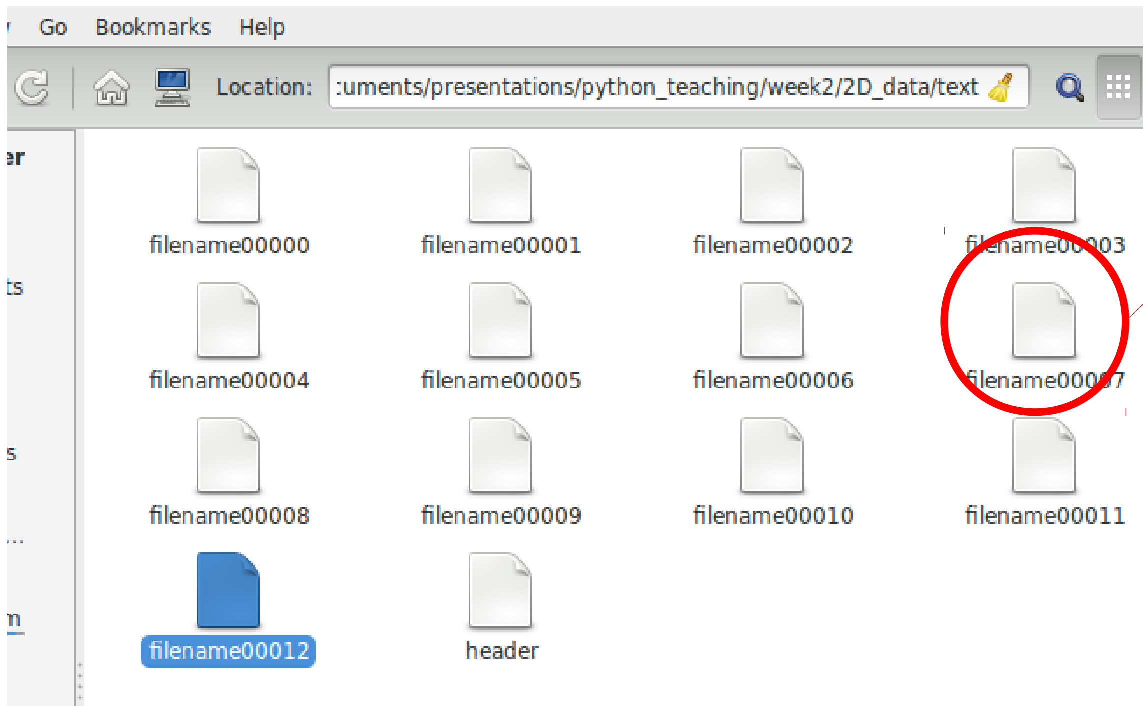


1.025468
2.0198
-3.2471
….

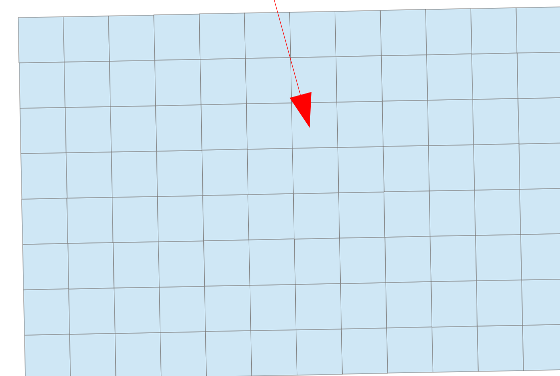


Three Sample Datatypes

- Choice of three formats of data for hands on exercise (or use your own)
  - Column - with a seperate header file
  - Binary - with seperate header (May not work for you, see Endianness)
  - Text - with included header file and other fields



The data in each file is 4200 floats describing a 84 by 50 2D field



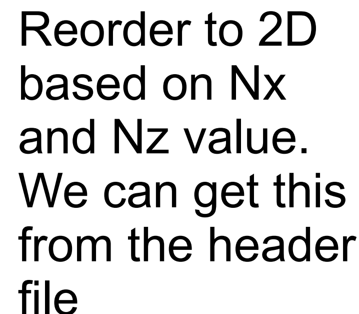
Reading 2D Data from a Column

- Reading data stored as a single column

```
# Numpy function to read column data

f = "./column/filename00001"

data = np.genfromtxt(f)

field = data.reshape(84, 50)
```

header file

Nx 84
Nz 50
Lx 1560.41523408474
Lz 1069.90830902657
Nrecs 12



Reading 2D Data from a Binary File

- Reading Binary Format data

```
# Numpy helper function to read binary

f = "./binary/filename00001"

data = np.fromfile(open(f ,'rb'), dtype='d')

field = data.reshape(84, 50)
```

header file

Nx 84
Nz 50
Lx 1560.41523408474
Lz 1069.90830902657
Nrecs 12

Reading 2D Data from a Formatted Output File

```
/*--------------------------------*- C++ -*----------------------------------*|
=========                 |                                                 |
| \      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \    /   O peration     | Version:  3.0.1                                 |
|   \  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \/     M anipulation  | NOTE - THIS IS A FAKE FILE FOR PYTHON TEACHING  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    Nx        84;
    Nz        50;
    Lx        1560.41523408474;
    Lz        1069.90830902657;
    Nrecs     12
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 3 -1 0 0 0 0];

internalField   nonuniform List<scalar>
4200
(
-0.0310257085323
-0.0625208281593
-0.0440674291947
```

Header information in file

84 times 50 = 4200 records between brackets

Develop Three Functions to Postprocess

```
import matplotlib.pyplot as plt
# .. FUNCTIONS DEFINED HERE ..
# 1) read_header, 2) get_files and 3) read_file
foldername = './column/'
header = read_header(foldername+'header')
files = get_files(foldername, filename='filename')
for f in files:
    data = read_file(f)
    field = data.reshape(header['Nx'],header['Nz'])
    plt.imshow(field)
    plt.colorbar()
    plt.show()
```

Develop Three Functions to Postprocess

```
header = read_header(foldername+'header')
```

- Input is the name of the folder which contains the header file
- Code should open the header file, read the data as a string and convert it so it is stored in a dictionary with variable names as keys for their associated values
- Required output is this dictionary

```
files = get_files(foldername, filename='filename')
```

- Input is the foldername and basename of the files (in out case all files are of the form filename00000, filename00001, etc
- Required output is a list of files

```
data = read_file(f)
```

- Input is a single filename to be read
- Output is the content of the file, 4200 floats, as a numpy array

Hands-On Session 6

This hands on session is open-ended. The aim is to write three functions that can read text based meta-data from a file (header), identify a list of files to be read and then read that data and plot. **Please work on your own examples if you'd prefer**

1) Choose an input type from 2D_data folder

   column easy

   binary intermediate

   text is complex

2) Write a function read_file with inputs foldername and filename which returns all data. Check data using plt.imshow(data.reshape(84,50))

3) Write a function get_header which has input of the header file into a dictionary

4) Write a function get_files to get a list of all 13 files containing filenames.000* in a given directory

Solution

```
import matplotlib.pyplot as plt
# .. FUNCTIONS DEFINED HERE ..
# 1) read_header, 2) get_files and 3) read_file
foldername = './column/'
header = read_header(foldername+'header')
files = get_files(foldername, filename='filename')
for f in files:
    data = read_file(f)
    field = data.reshape(header['Nx'],header['Nz'])
    plt.imshow(field)
    plt.colorbar()
    plt.show()
```

Best Practice in Designing a Python Project

Evolutions of a Python Project

- Python session to try things, copy to a simple script and test
- Group repeated code into functions to avoid repetition:
  - Reduces potential errors as less code to check
  - Improves readability as clear modular parts which can be tested
  - Easier to maintain and less to change as design evolves
- Collect together similar functions in a module
- **Group functions acting on an object into a class for that object**
- Utilise inheritance to further reduce code volume
- Create a package by adding __init__.py file to folder

Back to Post-Processing Example

All post processing functions for a particular foldername, we can group them in a class

```
import matplotlib.pyplot as plt

foldername = './column/'
header = read_header(foldername+'header')
files = get_files(foldername, filename='filename')
for f in files:
    data = read_file(f)
    field = data.reshape(header['Nx'],header['Nz'])
    plt.imshow(field)
    plt.colorbar()
    plt.show()
```

Classes for Post-Processing

- We can use classes with the data reading functions

```
class postproc():
    def __init__(self, foldername, headername, filename):
        self.foldername = foldername
        self.headername = headername
        self.filename = filename
    def get_header(self):
        f = open(self.foldername+self.headername)
        ...
    def get_files(self):
        ...
```

Classes for Post-Processing

- We can use classes with the data reading functions

```
class postproc():
    def __init__(self, foldername, headername, filename):
        self.foldername = foldername
        self.headername = headername
        self.filename = filename
        self.header = self.read_header()
        self.files = self.get_files()
    def get_header(self):
        f = open(self.foldername+self.headername)
        ...
    def get_files(self):
```

Classes for Post-Processing

- We can then use this as follows to get and plot data

```
pp = postproc(foldername='./binary/',
              headername='header',
              filename='filename')
for f in pp.files:
    data = pp.read_file(f)
    field = data.reshape(pp.header['Nx'],pp.header['Nz'])
    plt.imshow(field)
    plt.colorbar()
    plt.show()
```

Constructor:

Gets all possible files in folder
Reads all header information

Reader:

Load data as needed

Classes for Post-Processing

- We can then use this as follows to get and plot data

```
pp = postproc(foldername='./binary/',
              headername='header',
              filename='filename')
for f in in range(pp.get_Nrecs()):
    f = pp.read_field(i)
    plt.imshow(field)
    plt.colorbar()
    plt.show()
```

read_field

> Loads data using pp.read_file(f)

> Converts to 2D field with data.reshape(pp.header['Nx'], pp.header['Nz'])

Constructor:

Gets all possible files in folder
Reads all header information

Reader:

Load data as needed

Inheritance in Python

- A person can train in a particular area and gain specialist skills

```
class Person():

    def __init__(self, name, age):

        self.name = name

        self.age = age

    def say_name(self):

        print("Hello, I'm "

              + self.name)

bob = Artist('Bob Jones', 24)

jane = Scientist('Jane Bones', 32)

bob.say_name(); bob.do_art()

jane.say_name(); jane.do_science()
```

```
class Scientist(Person):
    def do_science(self):
        print(self.name +
              'is researching')
```

```
class Artist(Person):
    def do_art(self):
        print(self.name +
              'is painting')
```

A Hierarchy of Classes for Post-Processing

```
class postproc():
        …
    def read_file(self, filename):
        raise NotImplemented
        …
```

The base class defines the constructor, get_files, etc but does not specify how to read_files as this is unique to each data type

A Hierarchy of Classes for Postprocessing

```
class postproc():
        …
    def read_file(self, filename):
        raise NotImplemented
        …
```

The base class defines the constructor, get_files, etc but does not specify how to read_files as this is unique to each data type

```
#Binary IS A type of postproc reader
class postproc_binary(postproc):
    def read_file(self, filename):
        return np.fromfile(open(filename,'rb'), dtype='d')
class postproc_column(postproc):
    def read_file(self, filename):
        return np.genfromtxt(filename)
```

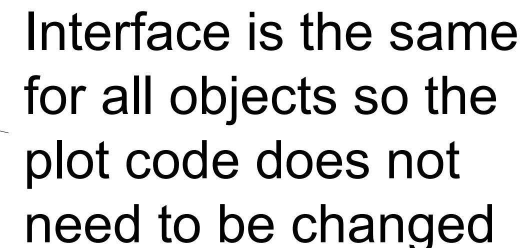
Inherit and only need to define read_file to customise for each data type

Text is a little more complex.... We need to redefine read_header as well

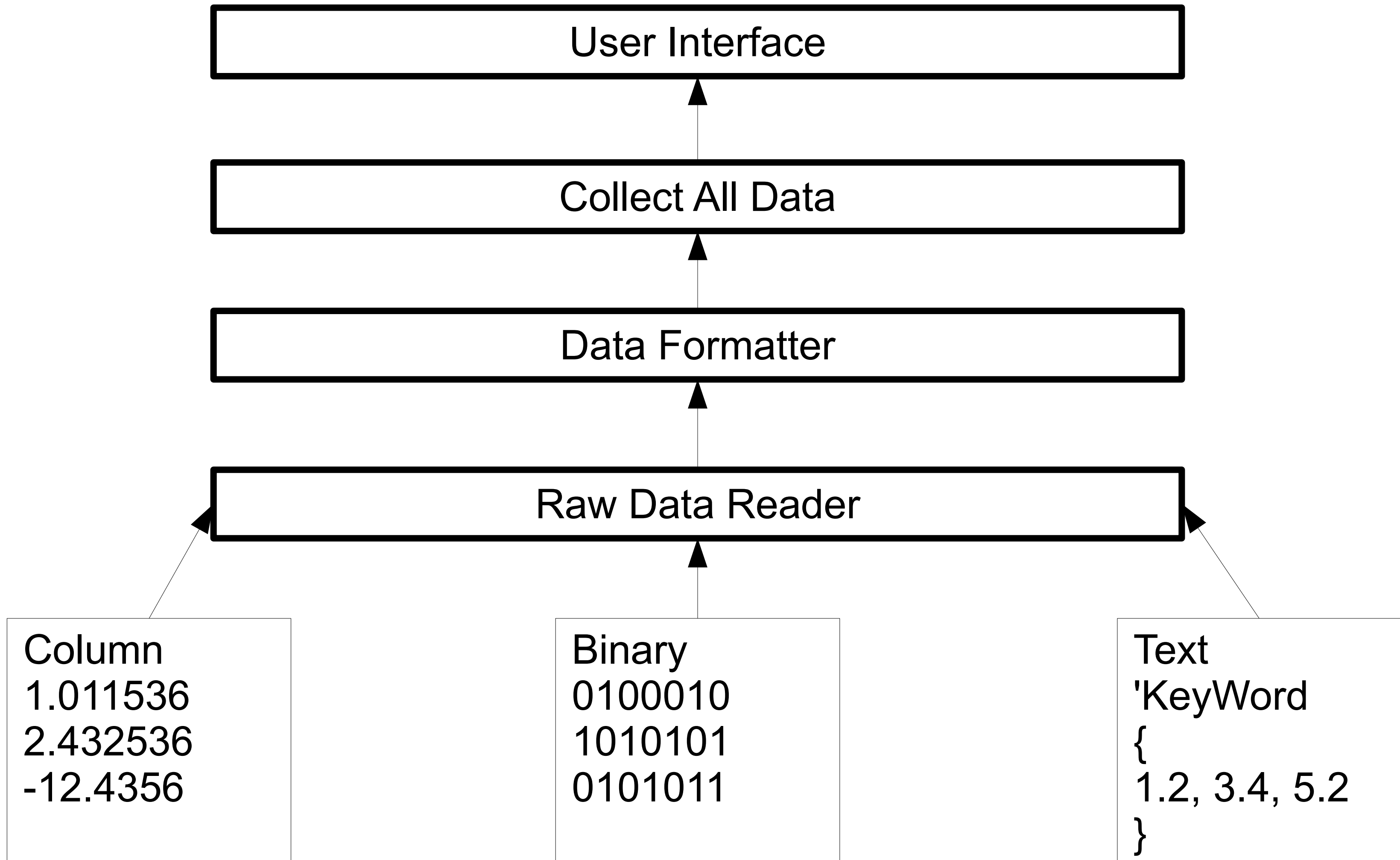
A Hierarchy of Classes for Post-Processing

- We can now plot any format of data

```
import postproclib as ppl

ds= "binary"
if ds is "text":
    pp = ppl.postproc_text(ds+'/', 'filename00000', 'filename')
elif ds is "column":
    pp = ppl.postproc_column(ds+'/', 'header', 'filename')
elif ds is "binary":
    pp = ppl.postproc_binary(ds+'/', 'header', 'filename')
print("Datasource is " + ds)
for i in range(pp.get_Nrecs()):
    f = pp.read_field(i)
    plt.imshow(f)
    plt.colorbar()
    plt.show()
```



Separate Data Reader Formatter and Plotter



Using Postproc Library with a Slider

```
import matplotlib.pyplot as plt
from matplotlib.widgets import
Slider
import postprocmod as ppl
#function which loads new
#record based on input
def update(i):
    print("record = ", int(i))
    field = pp.read_field(int(i))
    cm.set_array(field.ravel())
    plt.draw()
#Get postproc object and plot
initrec = 0
pp = ppl.postproc('./binary/',
         'header', 'filename')
field = pp.read_field(initrec)
cm = plt.pcolormesh(field)
plt.axis("tight")
```

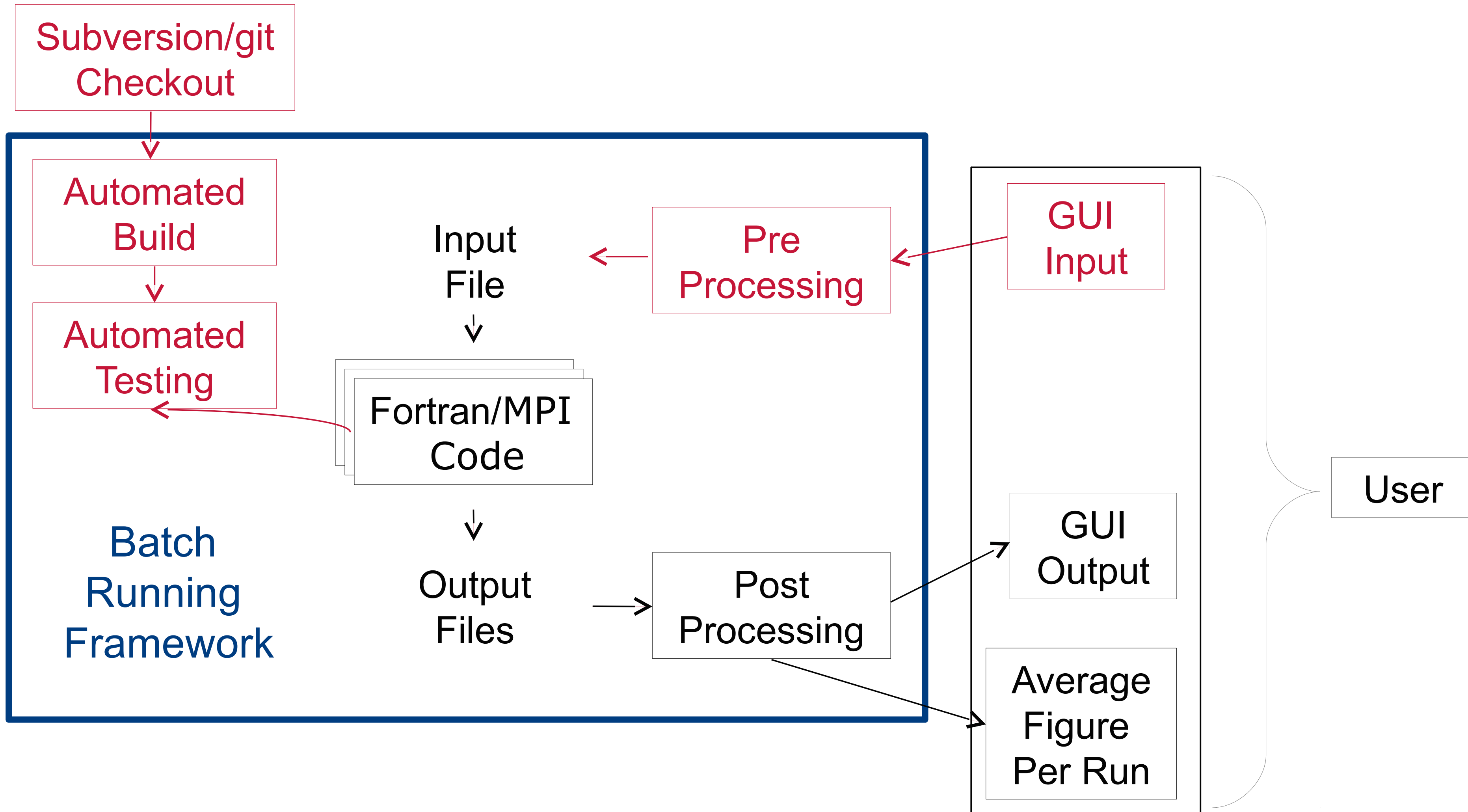
```
#Adjust figure to make room
for slider and add an axis
plt.subplots_adjust(bottom=0.2)
axslide = plt.axes(
    [0.15, 0.1, 0.75, 0.03])

#Bind update function
#to change in slider
s = Slider(axslide, 'Record',
       0, pp.get_Nrecs()-0.1,
       valinit=initrec)
s.on_changed(update)
plt.show()
```

A Similar Approach can Run Jobs Using Subprocess

- Use with class to wrap the build, setup, run and postprocess jobs

```
class Run():
    def __init__(self, rundir, srcdir):
        self.rd=rundir
        self.sd=srcdir
    def setup(self):
        #Create a copy of source code and compile
        shutil.copytree(self.sd, self.rd)
        subprocess.Popen("g++ " + self.rd + "test.cpp")
    def run(self):
        self.sp = subprocess.Popen(self.rd + "./a.out")
     def finish(self):
        files = pp.read_files(self.rd)
        for f in files:
            ...
```

How I use Python in my Work



Evolutions of a Python Project

- Python session to try things, copy to a simple script and test
- Group repeated code into functions to avoid repetition:
  - Reduces potential errors as less code to check
  - Improves readability as clear modular parts which can be tested
  - Easier to maintain and less to change as design evolves
- Collect together similar functions in a module
- Group functions acting on an object into a class for that object
- Utilise inheritance to further reduce code volume
- Create a package by adding __init__.py file to folder

Evolutions of a Python Project (Test Driven Development)

- Work out what you want the software to do
- Write tests first to define this desired functionality – Test Driven Development (TDD)
- Develop functions to pass these test scripts
- Collect together similar functions in a module
- Seperate tests into a suite – run test every time you make a change, or better still add to a continous intergration (CI) server
- Create a package by adding __init__.py file to folder
- Optionally refactor into a class and utilise inheritance to further reduce code volume (Design patterns may allow you to start OO)

The Function Interface

- The inputs to a function and returned output are like a contract with the user, 'give me this and I will give you that'
  - All three exercises returned the same data from different files
  - This means the same top level code could be used for any of the three data formats
  - This hides the form of the underlying data from the user, you only need to call read(filename) to get the data
- When releasing software, version number systems are based around this
  - From v1.0 to v1.1 the interface stays the same
  - If major number changes, e.g. v1.1 to v2.0, the interface has changed and is no longer backward compatible

Functional Interface

- Functions are like a contract with the user, here we take in the file name and return the data from the file

```
#Iterate through the files
for f in files:
    #read the data
    data = read_file(f)
```

TAKES A FILENAME AND RETURNS ITS CONTENTS

- We aim to design them so for a given input we get back the expected output

```
def square(a):
    return a**2
```

TAKES A NUMBER AND RETURNS ITS SQUARE

Unit Testing and TDD in Python

```
import unittest

def square(a):

    pass


class test_square(unittest.TestCase):

    def test_float(self):

        self.assertEqual(square(2.), 4.)

    def test_int(self):

        self.assertEqual(square(2), 4)

unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

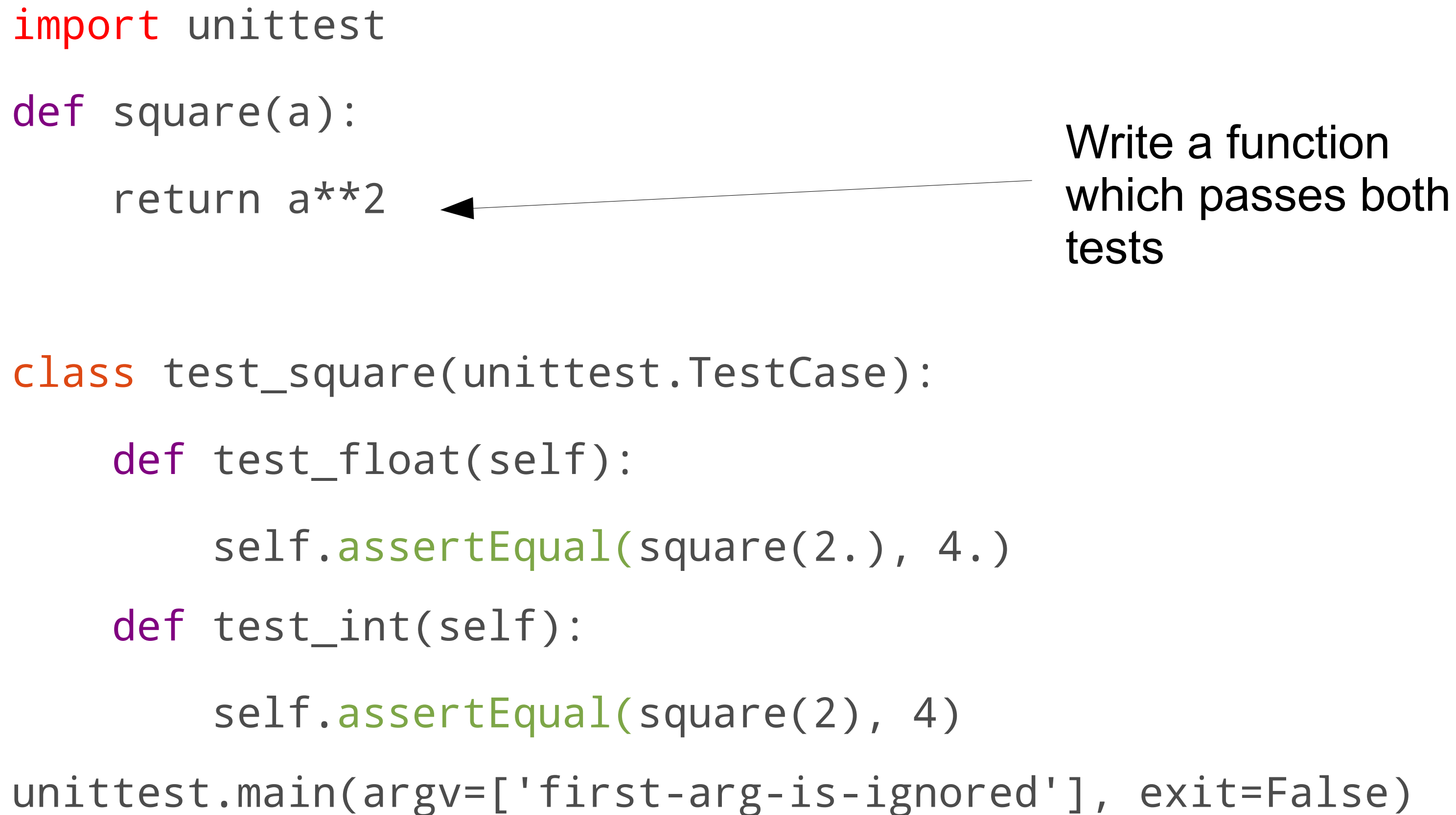
Function initial empty and written to satisfy required functionality

Required format for unittest (we'll review classes again soon)

Assert raises an error if the logical statement is not true

NOTE – Need arg-is-ignored to void an error in jupyter notebooks

Unit Testing and TDD in Python



NOTE – Need arg-is-ignored to void an error in jupyter notebooks

Unit Testing an Object

```
import unittest
class Number():
    def __init__(self, a):
        self.a = a
    def square(self):
        pass
    def cube(self):
        pass
```

Class methods empty and must be written to pass tests

Desired class functionality is DEFINED by the tests

```
class test_number(unittest.TestCase):
    def test_float(self):
        n = Number(2.)
        self.assertEqual(n.square(), 4.)
        self.assertEqual(n.cube(), 8.)
    def test_int(self):
        n = Number(2)
        self.assertEqual(n.square(), 4)
        self.assertEqual(n.cube(), 8)
unittest.main(argv=['first-arg-is-ignored'],
              exit=False)
```

Unit Testing an Object

```
import unittest
class Number():
    def __init__(self, a):
        self.a = a
    def square(self):
        return self.a**2
    def cube(self):
        return self.a**3
```

Class methods written in order to satisfy required functionality

Desired class functionality is DEFINED by the tests

```
class test_number(unittest.TestCase):
    def test_float(self):
        n = Number(2.)
        self.assertEqual(n.square(), 4.)
        self.assertEqual(n.cube(), 8.)
    def test_int(self):
        n = Number(2)
        self.assertEqual(n.square(), 4)
        self.assertEqual(n.cube(), 8)
unittest.main(argv=['first-arg-is-ignored'],
              exit=False)
```

Version Control

- Once you have some code, put it into a code repository
  - Backup in case you lose you computer
  - Access to code from home, work and anywhere else.
  - Allows you to keep a clear history of code changes
  - Only reasonable option when working together on a code
- Three main repositories are git, mercurial and subversion.
- Most common is git, a steep learing curve and helps the maintainer more than the developer (in my opinion). Mercurial may be better... Subversion is often disregarded due to centralised model.
- Range of free services for hosting, Imperial has a paid github account https://github.com/ so you can host close source projects

Automated Testing

- Travis CI – you write a script and your tests are run automatically. If your latest change breaks the test, you will get an email

```
os: linux
language: python
python:
    - 2.7
    - 3.6
script:
    - make test
```

- Or setup your own local solution using Python scripts
- **Start your test suite now!** It will improve your software development

Sample Project Online

https://github.com/edwardsmith999/python_example_project

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

edwardsmith999 committed on GitHub Update README.md — Latest commit 9a9a464 3 minutes ago

File	Commit message	Updated
.travis.yml	Removed Python 2.6 from tests	6 minutes ago
Makefile	Py.test causes weird import bug due to __init__ in folder, added make…	8 minutes ago
README.md	Update README.md	3 minutes ago
__init__.py	Minimal example of functions, Number class and travis test	16 minutes ago
number_class.py	Minimal example of functions, Number class and travis test	16 minutes ago
number_fns.py	Minimal example of functions, Number class and travis test	16 minutes ago
test_number_class.py	Minimal example of functions, Number class and travis test	16 minutes ago
test_number_fns.py	Minimal example of functions, Number class and travis test	16 minutes ago

README.md

python_example_project build passing

This project shows a minimal example of a Python module complete with Travis continous integration testing.

Hands-On Session 7

Introduction

1) Use test driven development (i.e. write the tests first) to design functions which returns the square and the cube of input values in a file called numbers.py.
2) Refactor numbers.py to a class with constructor to take input a, stores it as self.a = a and change the functions square and cube to act on self.a.
3) In numbers.py, add `if __name__ == "__main__":` and move the tests inside this. Use import numbers as nb in a new script, instantiate n = nb.number(5.) and use the n.square() and n.cube() methods.

Advanced

4) Refactor the three functions from the previous hands-on: **get_files, read_header** and **read_data** into a single class postproc. Write a constructor for the postproc class to take foldername, header and filename and get self.header and self.files.
5) Try steps 4) to 5) for a different input data format by creating a new class. Make postproc a base class and using Python inheritance syntax, e.g. `class postproc_binary(postproc):`, create a range of different data readers

What to do next?

- Find a project
  - Use Python instead of your desktop calculator
  - Ideally something at work and outside
- Use search engines for help, Python is ubiquitous - often you can find sample code and tutorials for exactly your problem
  - Stackoverflow is usually the best source of explanation
  - Official documentation is okay as a reference but not introductory, look for many excellent tutorials, guides and videos
  - help(function) in python. Tab, ? or ?? in ipython
- Be prepared for initial frustration!
  - Worth the effort to learn

What to do next?

- If we didn't cover something you needed for your work, please ask. I will also send notes to everyone who signed up.
- Please provide feedback on today
  - Was the course useful? What could be improved?
  - I believe Python should be taught at undergraduate level here at Imperial. Please support this by filling in the questionnaire, I will present the results on Wednesday HPC session (10 – 12 tomorrow)

http://bit.ly/2yf1vka

Or Link can be found here:

http://cpl-library.org/python_feedback.shtml