An introduction to Python for Scientific Computation

By Edward Smith

3rd March 2017
Aims for today

- Motivation for using Python.
- Introduction to basic syntax (lists, iterators, etc) and discussion of the differences to other languages.
- Scientific libraries numpy and matplotlib.
- Using python to read files (ASCII, CSV, Binary) and plot.
- Examples of usage for scientific problems.
Overview

- Introduction and Basic Constructs of Python (~30mins)
- Hands on session + break (~20 min)
- Programming in Python (~15 min)
- Hands on Session + break (~20 min)
- Scientific computing and data analysis (~20 min)
- Hands on session (~15 min)
Pros and Cons of Python (vs 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

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
Computing at Imperial

- Aeronautical Engineering – **MATLAB** in ”Computing” and ”Numerical Analysis”
- Bio-Engineering – **MATLAB** in ”Modelling for Biology”
- Chemical Engineering – Only **MATLAB** taught
- Chemistry – **Python** taught
- Civil Engineering – **MATLAB** in ”Computational Methods I and II” (some object oriented in second year)
- Computing/Electrical Engineering – low level
- Materials – **MATLAB** in ”Mathematics and Computing”
- Maths – **Python** in 2nd term (**MATLAB** in 1st)
- Mechanical Engineering – Only **MATLAB** taught
- Physics – Start 1st year ”Computing Labs” with **Python**
- Biology and Medicine – No programming?
My Background

• Currently a full time software developer/researcher
  – Civil Engineering (Prev Mech & Chem Eng at IC)
  – About 8 years of programming experience
  – Software Sustainability Fellow (www.software.ac.uk)
  – Answer Python questions on Stackoverflow
• Why this course?
  – I learnt MATLAB as undergrad in Mech Eng (also c++ and assembly language but still mainly used excel)
  – Masters project: Lattice Boltzmann solver in MATLAB. PhD: Fortran/MPI Molecular Dynamics, MATLAB post processing
  – Collaborator used Python and too much effort to maintain both but took me a year to kick the MATLAB habit
  – My main incentive for the switch to Python is the long term potential and the ability to write more sustainable code
  – I wish I had learnt Python sooner!
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
  - Tried to instantiate all possible field objects in a folder and plot
  - Based on wxpython and inspired by MATLAB sliceomatic

- Batch running framework for compiled code
  - Simple syntax for systematic changes to input files
  - Specify 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

Input File

Fortran/MPI Code

Output Files

Post Processing

GUI Output

User
How I use Python in my Work

- Input File
- Fortran/MPI Code
- Output Files
- Post Processing
- GUI Output
- Average Figure Per Run
- User

Batch Running Framework
How I use Python in my Work

Subversion/git Checkout

Automated Build

Automated Testing

Batch Running Framework

Input File

Fortran/MPI Code

Output Files

Pre Processing

Post Processing

GUI Input

GUI Output

Average Figure Per Run

User
Possible Future Extensions

- Subversion/git Checkout
- Automated Build
- Automated Testing
- Batch Running Framework

- Input File
- Fortran/MPI Code
- Output Files

- Pre Processing
- Machine Learning?
- Post Processing
- Results To Website

- Web Scraping
- GUI Input
- GUI Run Info
- GUI Output

- Average Figure Per Run

- User
Your Programming Experience

- No Experience
- Novice
- Intermediate
- Advanced
Python VS MATLAB (and R?)
An Example

%MATLAB

clear all
close all

x = linspace(0,2*pi,100);
y = sin(x);
z = cos(x);
plot(x,y,'-r');
hold all
plot(x,z,'-b')

#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, '-r')
plt.plot(x, z, '-b')
plt.show()
Some of your aims for the course

- Learn basics of python, ability to switch away from Matlab ...
- Improve my very basic knowledge of Python and understand the advantages of coding in general
- Introduction to Python-specifics (syntax, data types?, ...) rather than general programming concepts.
- Basic understanding of Python so that I can implement it with [PLUG IN] to speed up post-processing of data etc.
- Quite a few scripts that I am using for analysis … are written in Python so I would like to be able to understand better
- See what python could help me while doing research. Get the idea of Object Oriented Programming …
- Using python for Data processing and analysis. A general feel for other uses such as modelling.
- An understanding of a programming language that is currently in high demand.
Some of your aims for the course
My 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
My aims for the course

- 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
Key Concepts - Types

- Use the python command prompt as a calculator

3.141592653589*3.0  Out: 9.424777961
Key Concepts - Types

• Use the python command prompt as a calculator

3.141592653589*3.0 Out: 9.424777961

• Define variables as one of several types

a = 3.141592653589 # Float
i = 3 # Integer

Variables stay defined (Their “scope”) for duration of python session (or script).

Syntax here means: “Define a to be 3.141592653589” and “define i to be 3”
Key Concepts - Types

- Use the python command prompt as a calculator
  
  \[3.141592653589 \times 3.0\]  
  Out: 9.42477796076938

- Define variables as one of several types
  
  ```
  a = 3.141592653589  # Float
  i = 3  # Integer
  ```

- We can then perform the same calculations using variables
  
  ```
  a * i  Out: 9.42477796076938  # But float*integer
  ```

Variables stay defined (Their “scope”) for duration of python session (or script).
Key Concepts - Types

• Use the python command prompt as a calculator

```
3.141592653589*3.0    Out: 9.424777961
```

• Define variables as one of several types

```
a = 3.141592653589    # Float
i = 3                 # Integer
```

• We can then perform the same calculations using variables

```
a * i    Out: 9.4247796076938    # But float*integer
2 / i    Out: 0                 # WATCH OUT FOR int/int
2./ i    Out: 0.666666666666666 # Use floats for division
2/float(i) Out: 0.666666666666666 # Explicit conversion
histogram_entry = int(value/binsize) # Integer rounds up/down
```
Key Concepts - Functions

- Check type with
  
  type(a)  Out: float
  
  type(i)  Out: int

- type(), float() and int() are all examples of functions, i.e.
  - take some input,
  - perform some operation
  - return an output

\[ f(x) = x^2 \]

Input: \( x = 5.0 \)  \[ \rightarrow \]  Output: 25.0
Key Concepts - Functions

- Check type with
  
  ```
  type(a)  Out: float
  type(i)  Out: int
  ```

- `type()`, `float()` and `int()` are all examples of functions, i.e.
  - take some input,
  - perform some operation
  - return an output

```python
def square(input):
    """Function to calculate the square of a number""
    output = input*input
    return output
```

```
# Now we can use this square(5.0) Out: 25.0
```

\[ f(x) = x^2 \]

Input: \( x = 5.0 \)  
Output: 25.0

Note: indent whitespace instead of end
# Define a variable

```python
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

```python
# Define Function
def square(input):
    'calculate square'
    output = input * input
    return output
```

Document function here *"text"* for one line or **""" multi-line verbose and descriptive text """"**

Value to return from function

```python
# We call the function like this
square(a) Out: 25.0
```
Key Concepts - Functions

- Note that the input and output type are not specified

```python
def square(input):
    "calculate square"
    output = input*input
    return output

#Now we can use this
square(5.0) Out: 25.0
square(5) Out: 25
```

Key Concepts - Functions

$$f(x) = x^2$$

Input: $x = 5.0$

Output: 25.0
Key Concepts - Functions

- Note that the input and output type are not specified

```python
def square(input):
    "calculate square"
    output = input * input
    return output
```

# Now we can use this

```
square(5.0) Out: 25.0
square(5)   Out: 25
```

- Python allows "duck typing":
  - "If it looks like a duck and quacks like a duck, it's a duck"
  - Both useful and a possible source of error
  - `TypeError`: unsupported operand type(s)
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 redundant(a, b):
    return b

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

def quadratic(a, b, c):
    "Solve: y = ax^2 + 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

def get_27():
    return 27

#Call using
get_27()
```

Optional variable. Given a value if not specified.
Key Concepts - Functions

- Note that the input and output type are not specified

```
# Function to divide one number by another

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

# Which gives us
divide(2, 5) Out: 0
```

\[ f(x, y) = \frac{x}{y} \]
Key Concepts - Functions

- Note that the input and output type are not specified

#Function to divide one number by another
def divide(a, b):
    output = a/b
    return output

#Which gives us
divide(2, 5) Out: 0

#Maybe more sensible to define?
def divide(a, b):
    output = float(a)/float(b)
    return output
divide(2, 5) Out: 0.4

You can look at function information with:
help(square) in python
In ipython, also square? Or to see the code: square??
Key Concepts - Conditionals

- Allow logical tests

```python
# 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

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

Indent determine scope
4 spaces here
Key Concepts - Functions

- Note that the input and output type are not specified

```python
#Add a check
def divide(a, b):
    if ((type(a) is int) and (type(b) is int)):
        raise TypeError
    else:
        return a/b
```
• Note that the input and output type are not specified

```python
#Add a check
def divide(a, b):
    if ((type(a) is int) and 
        (type(b) is int)):
        raise TypeError
    else:
        return a/b
```

• Python error Handling – Better to ask forgiveness than seek permission

```python
try:
c = divide(a, b)
print(c)
except TypeError:
    print("Cannot divide a=", a, " by b=", b)
```
Part 1 Summary

- Two numerical types, floats and Integers
  
a = 2.5251  
i = 5  

- Functions allow set operations
  
def divide(a, b):
      output = a/b
      return output

- Conditional statement
  
if a > b:
    print(a)
elif a < b
    print(b)
else:
    print(a, b)

You can look at function information with:
help(type) in python
In ipython, also type? Or to see the code: type??

Some Functions

- type(in) – get type of in
- int(in), float(in) – Convert in to int, float
- help(in) – Get help on in

Design to prevent potential errors caused by Python's duck typing and lack of type checking
Hands on session 1 – Tutors

- Isaac and Edu

- 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 – Questions

• Introduction
  1) Get Python (ideally ipython) working... Please help each other here.
  2) Play around with basic arithmetic. Does this behave as expected? Note exceptions
  3) What does this do? i=3; i = i + 1
  4) Write a function to add two numbers and always return a float
  5) Use an if statement to print the larger of a or b
  6) Define a function to raise a floating point number to an integer power N. What changes would you need to make to raise to non-integer powers?

• More advanced
  1) Write a function which combines both 4) and 6) above to get the hypotenuse of a triangle from two side lengths $h^2 = o^2 + a^2$
  2) What does the function here do ===========>
  3) Write a recursive factorial function

```python
def add_fn(a, b, fn):
    return fn(a) + fn(b)
```
Key Concepts - Types

Define variables as one of several types

\[
a = 3.141592653589 \quad \# \text{ Float}
\]

\[
i = 3 \quad \# \text{ Integer}
\]

\[
s = "\text{some string}" \quad \# \text{ String}
\]
Strings

• String manipulations

s = "some string"

t = s + " with more"  Out: "some string with more"

s*3  Out: "some stringsome stringsome string"


s[0:4]  Out: some
Strings

- String manipulations

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

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

Note object oriented use of a function here. Instead of title(s) we have s.title(). The object s is automatically passed to the title function. A function in this form is called a method (c.f. c++ member function)
Strings

- Useful for opening and reading files (Saved as a string)

```python
# Get data from file

fdir = "C:/path/to/file/

f = open(fdir + './log')

filestr = f.read()

w = "keyword"

if w in filestr:
    indx = filestr.find(w)
    print(int(filestr[indx+len(w)+1]))

Out: 4
```

Note object oriented use of a function (method). Instead of read(f) we have f.read(). The object f is automatically passed to the read function.

All the contents of the file are read in as a string. This can be manipulated. E.g. if filestr = "contents of the file with some keyword=4 hidden inside"
Key Concepts - Types

Define variables as one of several types

a = 3.141592653589  # Float
i = 3            # Integer
s = "some string"  # String
l = [1,2,3]       # List, note square brackets tuple if ()
• Lists of integers

```python
l = [1, 2, 3]
l = l + [4]  # Note l.append(4) is an alternative
Out: [1, 2, 3, 4]
```

• We can make lists of any type

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

• But, these don't work in the same way as arrays

```python
l * 2  Out: [1, 2, 3, 4, 1, 2, 3, 4]
l * 2.0 Out: TypeError: can't multiply sequence by non-int of type 'float'
```
Loops or iterators

- Iterators – loop through the contents of a list

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

- Iterators – loop through the contents of a list
  
  ```python
  m = ['another string', 3, 3.141592653589793, [5,6]]
  for item in m:
      print(type(item), " with value ", item)
  ```

- Slightly more cumbersome for indexing
  
  ```python
  l = [1,2,3,4]
  for i in range(4):
      print("element", i, " is ", l[i] )
  ```

  \[\text{len(l) returns 4, range(4) returns a list with 4: [0,1,2,3]}\]
• Iterators – loop through the contents of a list

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

• Slightly more cumbersome for indexing

```python
l = [1,2,3,4]
for i in range(4):
    print("element", i, "is ", l[i])
```

• To add one to every element we could use

```python
for i in range(len(l)):
    l[i] = l[i] + 1
```
Loops or iterators

- Iterators – loop through the contents of a list

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

- Slightly more cumbersome for indexing

```python
l = [1,2,3,4]
for i in range(4):
    print("element", i, " is ", l[i] )
```

- To add one to every element we could use

```python
for i in range(len(l)):
    l[i] = l[i] + 1
```

Note: will not work:
```python
for i in l:
    i = i + 1
```

List comprehension
```python
l = [i+1 for i in l]
```
Define variables as one of several types

```python
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
```

---

**Key Concepts - Types**

Imperial College London

48
• Dictionaries for more complex data storage

```python
d = {"red":4, "blue":5}  #Dictionary
d["green"] = 6          #Adds an entry
print(d)
```

• Instead of numerical index, use a word to access elements

```python
print(d["red"])           
```

• Useful for building more complex data storage

```python
e = {"colours" : ["red", "blue"], "No": [3, 6]}
```

```
item  item
key   Value  key   Value  e.items()
```

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

- Dictionaries are for more complex data storage

```python
e = {"colours" : ["red", "blue"], "No": [3, 6]} #Dictionary
```

```python
e["colours"]    out: ["red", "blue"]
```

- Elements can also be accessed using key iterators

```python
for key in e.keys():
    print(key, e[key])
```

Out: ("colours", ["red", "blue"])

("No", [3, 6])
**Dictionaries**

- Could be used instead of variables, consider $F=ma$

  ```python
  Newton = {}
  Newton["F"] = 2.
  Newton["m"] = 0.5
  Newton["a"] = Newton["F"]/Newton["m"]
  ```

- More importantly, variables do not need to be known in advance

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

### log file ###

- Nsteps 1000
- domain_x 10.0
- domain_y 20.0
- timestep 0.5
- Nx 100
- Ny 200
Part 2 Summary

- **Strings**
  
  ```python
  s = "some string"
  t = s + " with more"
  ```

- **Lists and dictionaries**
  
  ```python
  m = ["another string", 3, 3.141592653589793, [5,6]]  # List
  d = {"red":4, "blue":5}  # Dictionary
  ```

- **Iterators (loops)**
  
  ```python
  for item in m:
      print(type(item), " with value ", item)
  # Loop with numbers

  for i in range(10):
      print(i)
  ```
Hands on session 2 – Questions

• Introduction

1) Build a sentence \( s \) by defining and adding the 4 strings "is", "a", "this" and "sentence" in the right order. Capitalise the first letter of each of the words. Print the first letter of each word. (note no unique way to do these).

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

3) Define two lists, one for odd and one for even numbers less than 10. Combine them to form a list of all numbers in the order \([1,2,3,4,5,6,7,8,9]\).

4) Using keys "even", "odd" and "combined" put lists from 3) in a single dictionary.

5) Using \( l = [1,2,3] \), write a loop to add a number to all elements giving \([2,3,4]\). Write a function to take in a list \( l \) and number \( N \), which adds \( N \) to all elements of \( l \).

• More advanced

1) For the string \( s="\text{test}" \) and the list \( l = ['t','e','s','t'] \), we see \( s[0] == l[0] \), \( s[1] == l[1] \). Are they equal? Can you convert the string to a list? What about list to string?

2) Define \( a = [1,2,3]; b = a; b.append(4) \). Why does \( a = [1,2,3,4] \)?

   what about if you use \( b = b + [4] \) instead of append?
Numerical and Plotting Libraries

- Numpy – The basis for all other numerical packages to allow arrays instead of lists (implemented in c so more efficient)
  - $x = \text{np.array([[1,2,3],[4,5,6],[7,8,9]])}$
  - mean, std, linspace, sin, cos, pi, etc
- Matplotlib – similar plotting functionality to MATLAB
  - plot, scatter, hist, bar, contourf, imagesc (imshow), etc
- Scipy
  - Replaces lots of the MATLAB toolboxes with optimisation, curve fitting, regression, etc
- Pandas
  - Dataframes to organise, perform statistics and plot data

NOTE: Downloading and installing packages is trivial with “pip” or conda
Numpy – The basis for all other numerical packages to allow arrays instead of lists (implemented in C so more efficient)

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

Dot means use array from module numpy. The numpy module is just a big collection of Python code where array (and many other things) are defined.

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

Use tab in ipython to see what code is available (or look online)
Define variables as one of several types

\begin{verbatim}
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
\end{verbatim}
Key Concepts – Arrays of data

- Lists of lists seem similar to matrices or arrays.

\[
m = \begin{bmatrix}
[1,2,3], [4,5,6], [7,8,9]
\end{bmatrix}
\]

\[
m[0][1] \quad \text{Out: 2}
\]

\[
m[1][2] \quad \text{Out: 6}
\]
Key Concepts – Arrays of data

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

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

- Lists are dynamic, you can add values and mix datatypes
Lists of lists seem similar to matrices or arrays. They are not!

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

- Lists are dynamic, you can add values and mix datatypes.
- For numerics, use Numpy arrays which are contiguous memory implemented in C (more efficient).

```python
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```
• Numpy – the basis for most numerical work (implemented in c so more efficient). Should be all the same type
  
  ```python
  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 (equiv to MATLAB x .* y)
  np.dot(x,y)  # Matrix multiply
  # Invert matrix using linear algebra submodule of numpy
  invy = np.linalg.inv(y)
  ```

• Numpy has a wide range of functions. As it is written in c, it is often faster to perform operations with numpy instead of loops
**Key Concepts – Arrays of data**

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

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

- Numpy allows statistical operations

  ```python
  x.mean()        Out: 5.0       (Note np.mean(x) equivalent)
  x.std()         Out: 2.5819888974716112  (Also np.std(x))
  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)
  ```

Method to get shape returns 2 elements for a 2D array, accessed by index
Importing Numerical and Plotting Libraries

- matplotlib – similar plotting functionality to MATLAB

```python
import matplotlib.pyplot as plt
plt.plot(x)
plt.show()
```

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 vs MATLAB

%MATLAB

clear all

close all

x = linspace(0,2*pi,100);
y = sin(x);
z = cos(x);
plot(x,y,'-r');
hold all
plot(x,z,'-b')

#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,"-r")
plt.plot(x,z,"-b")
plt.show()
An Example vs MATLAB

%MATLAB

clear all

close all

x = linspace(0,2*pi,100);
y = sin(x);
z = cos(x);

plot(x,y,'-r');

hold all

plot(x,z,'-b')

show()

Better not to do this to avoid nameclashes

#python

from numpy import *

from matplotlib.pyplot import *

x = linspace(0,2*pi,100)
y = sin(x)
z = cos(x)

plot(x,y,'-r')

plot(x,z,'-b')

show()
An Example plotting a histogram

```python
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()
```
An Example plotting a 2D field (matrix)

```python
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 plotting a 2D field + function + loop

```python
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 using time series

```python
plt.ioff()
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 using data from a csv file

```python
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()
```

<table>
<thead>
<tr>
<th>file.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y</td>
</tr>
<tr>
<td>1.0, 1.0</td>
</tr>
<tr>
<td>2.0, 4.0</td>
</tr>
<tr>
<td>3.0, 9.0</td>
</tr>
<tr>
<td>4.0, 16.0</td>
</tr>
<tr>
<td>5.0, 25.0</td>
</tr>
<tr>
<td>6.0, 36.0</td>
</tr>
</tbody>
</table>
An Example using data from a csv file + function

```python
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
cube.csv
x, y
1.0, 1.0 1.0, 1.0
2.0, 4.0 2.0, 8.0
3.0, 9.0 3.0, 27.0
4.0, 16.0 4.0, 64.0
5.0, 25.0 5.0, 125.0
6.0, 36.0 6.0, 216.0
```
Reading from files

• Opening and finding keywords in file

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

• Reading binary data (see e.g. stackoverflow)

```python
with open('./log.bin', 'rb') as f:
    filecontent = f.read()
    struct.unpack("iii", filecontent)  #Need to import struct
```
Overview

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

- Graphical User Interfaces (GUI) e.g. Tkinter, wxpython, pyGTK, pyQT
- Multi-threading and parallel e.g. Subprocess, MPI
- Image and video manipulation e.g. pyCV, PIL
- Machine learning e.g. Scikit-learn, Pybrain
- Build system e.g. scons, make using os/system
- Differential equations solvers e.g. FEniCS, Firedrake
- Databasing and file storage e.g. h5py, pysqlite
- Web and networking e.g. HTTPLib2, twisted, django, flask
- Web scraping – e.g. scrapy, beautiful soup
- Any many others, e.g. PyGame, maps, audio, cryptography, etc, etc
- Wrappers/Glue for accelerated code e.g. HOOMD, PyFR (CUDA)
- It is also possible to roll your own
Summary

- **Background and motivations for this talk**
  - MATLAB is the main programming language taught at Imperial
  - Python provides similar plotting, numerical analysis and more

- **Some key concepts**
  - Data types, lists/arrays, conditionals, iterators and functions
  - Modules for scientific computing: numpy and matplotlib
  - Clean syntax, ease of use but no checking!

- **Advantages of learning Python**
  - General programming (better both in academia and outside)
  - Allows an integrated framework and can be bundled with code
  - Open source libraries with tutorials and excellent help online
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 so often you can find sample code and tutorials for exactly your problem
  - Stackoverflow is often 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
Next Week

- Next Friday 10th March, 2017 14:15-16:15 SAF 120
- Further details of the Python language
  a) More on Python data structures.
  b) Use of functions and design of interfaces.
  c) Introduction to classes and objects.
  d) Structuring a project, importing modules and writing tests.
  e) Examples of usage for scientific problems.
- Please provide feedback on the quadratics form (link on your email) to help tailor the course
Hands on session 3 – Questions

• Introduction
  1) Import numpy and matplotlib. These may not be installed so you will need to use conda, pip, easy_install or some other means of getting them.
  2) 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.
  3) Plot a tanh function in the range -2 \( \pi \) to 2 \( \pi \) using linspace and matplotlib plot.
  4) Create a 1D array of 10,000 normally distributed random numbers t. Plot as a time history and zoom in to see the detail.
  5) Plot a histogram of the array t from question 4) with 50 bins.
  6) Convert array t to a 2D array using t.reshape(100,100) and plot using contourf.
  7) Create a comma separated variable file (e.g. 2 columns in excel). Import into Python using np.genfromtxt("./file.csv", delimiter=','\) and plot. Check against the plot from excel or other software.

• More Advanced/open ended
  – Apply python to read some data from your research. Use numpy to perform basic statistical tests (results as expected). Plot using matplotlib.