Python II

Dr. John Quinn

Modules

- In Python, modules are collections of functions (and classes) grouped together that have some common use.
- There are various ways to 'load' the module or functions in it: (for example to use the function `linspace()` from the module `numpy`:

`import numpy`	`numpy.linspace(…)`	loads the entire numpy module, access via `numpy.fnc_name()`
`from numpy import *`	`linspace(…)`	BAD! Loads the entire numpy module but names may clash with other names. Not clear which module `linspace()` is from
`import numpy as np`	`np.linspace(…)`	load the entire numpy module, access via `np.fnc_name()`
`from numpy import linspace`	`linspace(…)`	only imports the single numpy function `linspace()`.
`from numpy import linspace as ls`	`ls(…)`	BAD! Works but causes confusion

Functions

- It is very easy to make one's own functions in Python.
- The general format is:

```
def functionname( parameters ):
   "optional function_docstring"
   function_code
   return [expression] % return value optional
```

- To load a function:
  - have it in your script file before you call it.
  - have it in its own file and then `import` it (the function `square( )` will be in a module called `square` so call with `square.square( )`.

3

Functions

- Example: function typed into iPython interpreter/notebook

```
def square(x):
    """Returns the square of a number, passed as the only argument"""
    sqr=x**2
    return sqr
```

```
In [23]: help(square)
Help on function square in module __main__:

square(x)
    Returns the square of a number, passed as the only argument

In [24]: square(4)
Out[24]: 16

In [25]: x=[square(x) for x in range(10)]

In [26]: x
Out[26]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Functions

- Example: function in its own script file ("square.py", in current directory)

```
In [4]: import square

In [5]: help(square)
Help on module square:

NAME
    square - Created on Wed Sep 10 08:11:03 2014

DESCRIPTION
    @author: quinn

FUNCTIONS
    square(x)
        Returns the square of a number, passed as the only argument

FILE
    /Users/quinn/temp/square.py

In [6]: square.square(5)
Out[6]: 25
```

Functions

- Functions can take multiple arguments, optional arguments etc. See documentation.
- Functions are “pass by reference”, meaning they can modify the values of variables passed to them as arguments (if they are mutable) and the changes remain once the function exits.

Useful Python Modules

`math`	core Python math module: see: https://docs.python.org/3/library/math.html
`numpy`	Numerical Python - MATLAB-like, vectorised operations, has its own math functions, fitting etc.
`scipy`	vast library of scientific functions, symbolic math etc.
`matplotlib`	2D and 3D plotting, publication-quality plots, highly configurable

Anaconda comes with 270+ packages: http://docs.continuum.io/anaconda/pkg-docs.html

Scipy, Numpy, Matplotlib

- Main Reference
  - http://www.scipy.org
- Fantastic online tutorials:
  - http://scipy-lectures.github.io

Numpy Essentials

- Recommend: `import numpy as np,` assumed for following examples
- Good Guide: http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf
- Cheat Sheets:
  - https://www.dataquest.io/blog/numpy-cheat-sheet/
  - https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

Numpy Essentials

- Numpy arrays are of fixed size! Extending means copying (poor performance)
- Creating arrays (not exhaustive)

```
a = np.array([1, 2, 3])   # int64

a = np.array([1., 2, 3])  # float64

a = np.array([1, 2, 3], float)  #explicitly specify type

a = np.array(range(10), float)

x=[1, 2, 3, 4, 5]
a=np.array(x,float)

a=np.array(range(10),float)
b=a                           # b and a are the same array -
                              # changing b changes a
                              # can check with "b is a"

c=a.copy()                    # c is a copy of a, or use c=a[:]

a.dtype   # tells you the type
```

Caution!

```
In [29]: x=np.arange(10)

In [30]: x
Out[30]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [31]: x[0]=10.5

In [32]: x
Out[32]: array([10,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

- No warning/error that float has been truncated!
- Solutions: explicitly make array as float type:

```
In [33]: x=np.arange(10.0)

In [34]: x
Out[34]: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

```
In [35]: x=np.arange(10,dtype=float)

In [36]: x
Out[36]: array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

Numpy

- Some useful functions for creating arrays (not exhaustive!):

```
ones()          % an array filled with 1s

zeros()         % an array filled with 0s

arange()        % evenly spaced values in a given interval [start,stop)
                  (last pt. not included)
                  Usage: numpy.arange([start, ]stop, [step, ]dtype=None)
                  start, and step are optional and default to 0 and 1
                  respectively

linspace()      % evenly spaced values in range [start,stop] (end pt. included)
                  Usage: numpy.linspace(start, stop, num=50,…)
                  Number of points are specified but can be changed with
                  another option (not shown above - see help)

logspace()      % evenly spaced on a log scale. Note start and
                  stop values should be the log of the start and
                  top you want as it generates in the interval
                  [start**10, stop**10]
```

Numpy

- 2D arrays

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
>>> a.flatten()
array([ 1.,  2.,  3.,  4.,  5.,  6.])
```

```
>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
>>> a.transpose()          # copy of a, transposed
array([[ 0.,  3.],
       [ 1.,  4.],
       [ 2.,  5.]])

>>> a.shape
(2,3)
```

Note: an array of size 6 is different from a 2d array of size (1,6): cf `newaxis`

Matrices

- Note: Numpy also has matrices for doing matrix operations.
- In some cases Matrices are to be preferred and the notation for multiplication etc is much cleaner.
- Not covered in this lecture.

Python >= 3.5 and NumPy has support for Matrix notation on 2D arrays!

Numpy

- Accessing elements: use Python's slice notation (1D):

```
>>> a = np.array([1, 4, 5, 8], float)

>>> a[:2]
array([ 1.,  4.])

>>> a[3]
8.0

>>> a[0] = 5.
>>> a
array([ 5.,  4.,  5.,  8.])
```

Numpy

- Accessing elements: use Python's slice notation (2D)

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> a
array([[ 1,  2,  3],
       [ 4,  5,  6]])

>>> a[0,0]
1

>>> a[0,1]
2

>>> a[1,:]
array([ 4,  5,  6])

>>> a[:,2]
array([ 3,  6])

>>> a[-1:,-2:]
array([[ 5,  6]])
```

Numpy Essentials

- (A subset of) operations on arrays:

```
np.sum()
np.min()
np.max()
:

np.average()
np.mean()
np.std()
np.var()
np.cov()
:

np.correlate()
:

np.histogram()
:
```

- Clever logical comparisons can be done element-by-element between arrays.
- Clever indexing can also be done - please see the documentation (in particular the `where()` function).

Numpy Essentials

- There are functions to extend, insert elements and delete elements:
  - `np.append()`
  - `np.insert()`
  - `np.delete()`

- All return copies of the original arrays with the requested modifications.
- See help for syntax.

Numpy Essentials

- Random Numbers: (`np.random`):
  - http://docs.scipy.org/doc/numpy/reference/routines.random.html

```
>>> np.random.rand()
0.4672448604689683

>>> np.random.rand(5)
array([ 0.52193671,  0.11871576,  0.0208547 ,  0.92195409,  0.08870152])

>>> np.random.rand(3,2)
array([[ 0.61778805,  0.55950128],
       [ 0.58063016,  0.68191155],
       [ 0.26972569,  0.31129301]])
```

Numpy Math

- Numpy has its own math routines that you should use with numpy arrays: http://docs.scipy.org/doc/numpy/reference/routines.math.html
- All operations are vectorised, i.e. they operate on all elements of the array.
- Example:

```
>>> x=np.linspace(0,10,10)

>>> y=np.sin(x)
```


20

Numpy Essentials

- Numpy is compiled C code and should be much faster than the Python equivalent.

```
>>> import math


>>> x=list(range(1000000))
>>> x=[float(a) for a in x]

%timeit [math.cos(a) for a in x]
10 loops, best of 3: 179 ms per loop


>>> y=np.array(x,float)

%timeit np.cos(y)
10 loops, best of 3: 20.3 ms per loop
```

Numpy Essentials

- Use numpy's `loadtxt()` function to read data from a file:
  - it automatically ignores lines starting with a # so you can include comments at the start of your file.
  - it loads the data into an array:

Here is the file, called "data.txt":

```
# data.txt
# columns are x, y, error
1 2.0 0.2
2 2.5 0.1
3 2.6 0.15
4 3.1 0.2
5 3.8 0.3
6 3.7 0.2
7 4.2 0.2
8 4.5 0.2
9 5.5 0.3
10 6.1 0.5
```

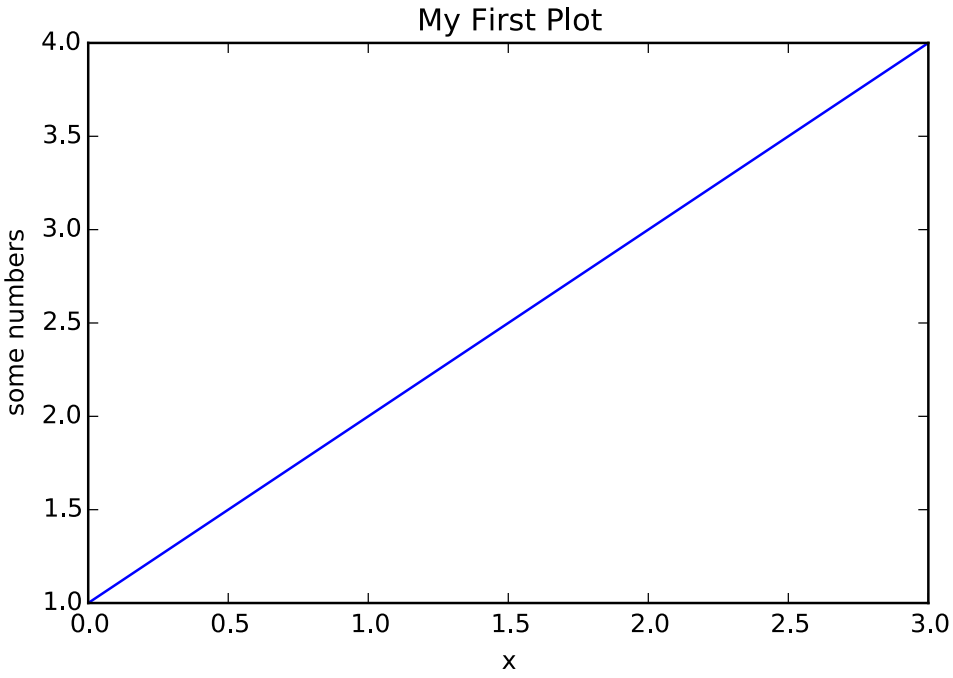
```
>>> data=np.loadtxt("data.txt")
>>> data
array([[  1.  ,   2.  ,   0.2 ],
       [  2.  ,   2.5 ,   0.1 ],
       [  3.  ,   2.6 ,   0.15],
       [  4.  ,   3.1 ,   0.2 ],
       [  5.  ,   3.8 ,   0.3 ],
       [  6.  ,   3.7 ,   0.2 ],
       [  7.  ,   4.2 ,   0.2 ],
       [  8.  ,   4.5 ,   0.2 ],
       [  9.  ,   5.5 ,   0.3 ],
       [ 10.  ,   6.1 ,   0.5 ]])

>>> x=data[:,0]
>>> y=data[:,1]
>>> err=data[:,2]
```


22

Matplotlib Essentials

- Matplotlib is a very powerful plotting package with lots of features and is very customisable.
- We are going to be concerned only with the `pyplot` module in `matplotlib`:
  - tutorial at: http://matplotlib.org/users/pyplot_tutorial.html
  - note that any figure can be saved with: `savefig()`, e.g.

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.xlabel('x')
plt.title('My First Plot')
plt.show()
plt.savefig("plot.pdf")
```

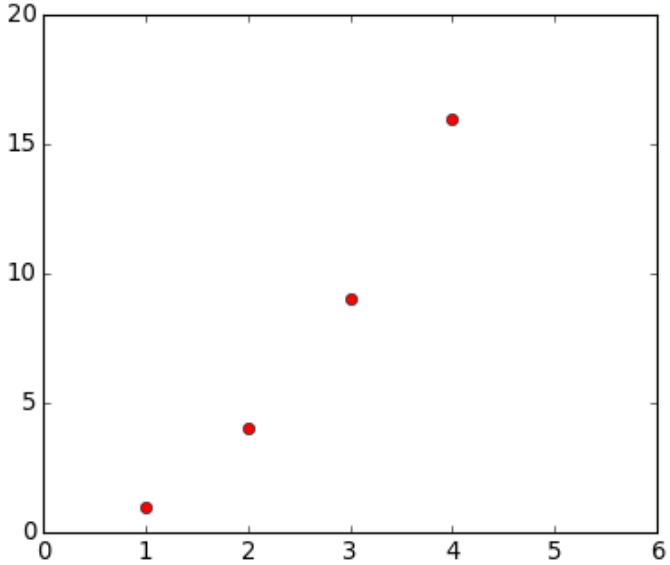


Format from name extension.
Allowed: png, jpg, pdf, ps, eps, svg amongst those allowed.

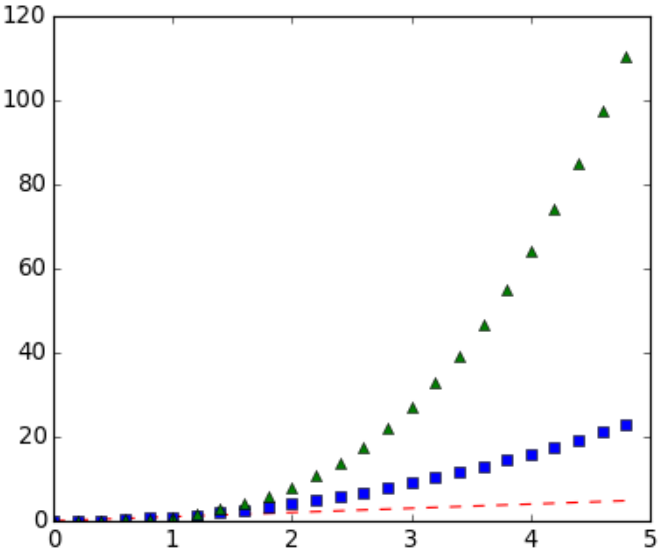
23

Matplotlib Essentials

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



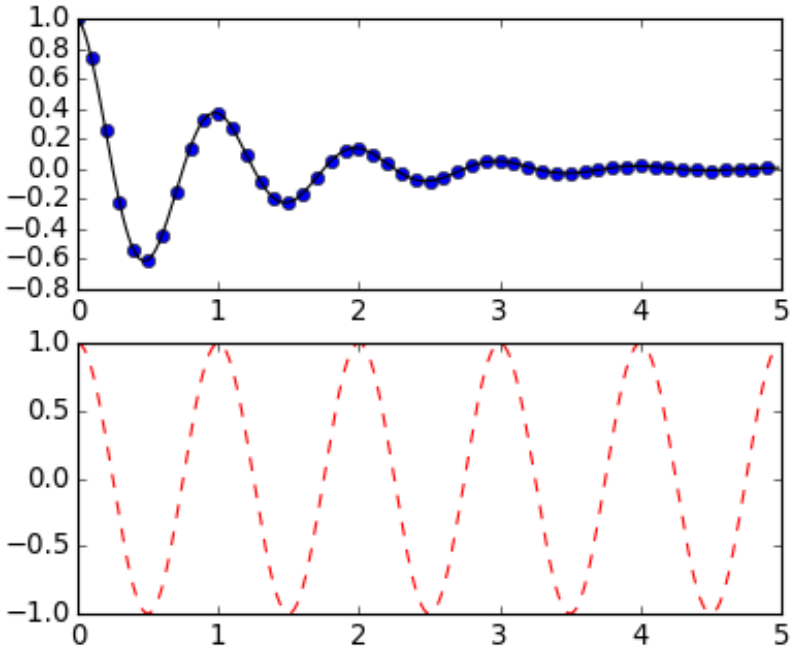

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
y1=t
y2=t**2
y3=t**3

# red dashes, blue squares and green triangles
plt.plot(t, y1, 'r--', t, y2, 'bs', t, y3, 'g^')
plt.show()
```



24

Matplotlib Essentials

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

t1 = np.linspace(0.0, 5.0, 50)
t2 = np.linspace(0.0, 5.0, 500)

y1= np.exp(-t1) * np.cos(2*np.pi*t1)
y2= np.exp(-t2) * np.cos(2*np.pi*t2)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, y1, 'bo', t2, y2, 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



25

Jupyter Notebook

- Consist of cells.
- Cells can be Python code or Markdown cells (e.g. see http://daringfireball.net/projects/markdown/)
  - Code cells can be run individually.
  - Markdown cells can include Latex for equations and html.
- Matplotlib Plots can be included in the notebooks, to embed plots use one of:
  - `%matplotlib inline` (static plots)
  - `%matplotlib notetebook` (interactive plots - new and buggy)
  - `plt.show()` is not needed in the notebook to produce plots.
- Notebooks can be exported as python, html, latex, or pdf (via latex)
- Many Python packages interact with IPython Notebooks to produce nicely formatted output (e.g. sympy)
- Github will render notebooks as html.
- Mac OS X Jupyter Notebook Viewer/ Quicklook: https://github.com/tuxu/nbviewer-app

26

Jupyter Notebook Widgets

- https://ipywidgets.readthedocs.io/en/stable/
- Useful e.g. if a long-running program:

In [1]:
```
from time import sleep
from numpy.random import rand
```

In [2]:
```
import ipywidgets as widgets
from IPython.display import display
```

In [3]:
```
p=widgets.FloatProgress(min=0, max=100)
display(p)
```

In [5]:
```
p.value=0
for i in range(100):
    sleep(rand()/10)
    p.value+=1
```