# Phys 60441
# Techniques of Radio Astronomy
# Part 1: Python Programming
# LECTURE 3

Tim O'Brien

Room 3.214 Alan Turing Building

tim.obrien@manchester.ac.uk

# Tuples

- Lists and strings are examples of sequences.
- A tuple is also a sequence of elements separated by commas and which can be enclosed by parentheses (round brackets) but need not be.

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
>>> t
>>> u = t, (1, 2, 3, 4, 5)
>>> u
```

Tuples may be nested (need to specify brackets here to make it clear)

- BUT elements of a list ***can*** be changed (mutable), elements of a string and a tuple ***cannot*** be changed.
- Tuples are useful for storing (x,y) coordinate pairs

# Dictionaries

- Sequences are indexed by numbers i.e. $0^{th}$ element, $1^{st}$ element etc.
- Dictionaries are indexed by *keys,* (which can be any immutable element e.g. strings, numbers, tuples containing immutable elements
- A dictionary is an unordered set of *key : value* pairs enclosed in curly brackets, with each key unique

```
>>> diameter = {‘Lov’ : 76, ‘Cam’ : 32, ‘Pic’ : 25}
>>> diameter[‘Are’] = 305
>>> diameter[‘Pic’]
>>> diameter
>>> del diameter[‘Are’]
>>> diameter.keys()
>>> ‘Cam’ in diameter
>>> for t, d in diameter.iteritems():
...      print t, d
...
```

Like using enumerate() function in a sequence

"Num-Pie"
http://numpy.scipy.org

"Sci-Pie"
http://www.scipy.org

http://matplotlib.sourceforge.net/

- **Python** is a general purpose programming language. It is interpreted and dynamically typed and is very suited for interactive work and quick prototyping, while being powerful enough to write large applications in.
- **NumPy** is a Python extension module, written mostly in C, that defines the numerical array and matrix types and basic operations on them.
- **SciPy** is another Python library that uses NumPy to do advanced maths, signal processing, optimization, statistics and much more.
- **matplotlib** is a Python library that facilitates publication-quality interactive plotting.

# NumPy - `ndarray`

To use NumPy, SciPy, matplotlib you need to connect to one of cosmogp2, cosmogp3... Cosmogp6 (ssh –X cosmogp2)

- NumPy's main object is the homogeneous multidimensional array called `ndarray`.
  - This is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Typical examples of multidimensional arrays include vectors, matrices, images and spreadsheets.
  - Dimensions usually called axes, number of axes is the rank

| | |
|---|---|
| [7, 5, -1] | An array of rank 1 i.e. It has 1 axis of length 3 |
| [ [ 1.5, 0.2, -3.7] , [ 0.1, 1.7, 2.9] ] | An array of rank 2 i.e. It has 2 axes, the first length 3, the second of length 3 (a matrix with 2 rows and 3 columns) |

# `ndarray` attributes

- **ndarray.ndim**
  - the number of axes (dimensions) of the array i.e. the *rank*.
- **ndarray.shape**
  - the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n* rows and *m* columns, shape will be (n,m). The length of the shape tuple is therefore the rank, or number of dimensions, ndim.
- **ndarray.size**
  - the total number of elements of the array, equal to the product of the elements of shape.
- **ndarray.dtype**
  - an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. NumPy provides many, for example bool_, character, int_, int8, int16, int32, int64, float_, float8, float16, float32, float64, complex_, complex64, object_.
- **ndarray.itemsize**
  - the size in bytes of each element of the array. E.g. for elements of type float64, itemsize is 8 (=64/8), while complex32 has itemsize 4 (=32/8) (equivalent to ndarray.dtype.itemsize).
- **ndarray.data**
  - the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

# Examples of `ndarray`

```
>>> import numpy
>>> a = numpy.arange(10).reshape(2,5)
>>> a
```

Creates an array with element values 0 to 9, reshaped to have 2 rows and 5 columns.

Try typing a.shape, a.ndim, a.size, a.dtype.name, a.itemsize

```
>>> import numpy
>>> b = numpy.array ( [ [1.5, 2., 3], [4, 5, 6] ] )
```

Use array function to create arrays. This is an array of floats.

```
>>> from numpy import *
>>> c = array( [ [1,2], [3,4] ], dtype=complex )
```

Note way of calling array depends on how numpy is imported

Can explicitly define type of array

```
>>> d = array( 1,2,3,4)
```

ERROR! Need square brackets to define a *list* of elements.

```
>>> zeros ( (3,4) )
>>> ones ( (2,3,4) )
>>> arange ( 1, 10, 2 )  # range from 1 to 10 spaced by 2
>>> linspace ( 0, 2*pi, 100 ) # 100 numbers from 0 to 2π
```

There are several useful ways of creating special arrays.

# Arithmetic operations

- Operations apply element by element

```
>>> a = array ( [ [2,2], [2,2] ] )
>>> b = ones( (2,2) )
>>> a+b
>>> 3*b
>>> a*b
>>> dot(a,b)   # also use mat(a)*mat(b)
>>> a *= 2
>>> b += 10
```

Careful! * is not a matrix product

a *=2   means   a = 2*a

```
>>> a = random.random ( (2,3) )
>>> a.sum()
>>> a.min()
>>> a.max(axis=1)  # max of each row
```

There are lots of other functions that apply to ndarrays.

# Indexing, slicing & iterating

- Just like with lists and other sequences (e.g. strings)

```
>>> a = random.random ( (10) )
>>> a[0]      #   first element
>>> a[2:5]    #  3rd element to 5th element
>>> b = random.random ( (4,5) )
>>> b[0]      #  first row
>>> b[2,3]    #  element in 3rd row, 4th column
>>>  b[: , 1]  #  2nd column
>>> for row in b:
...       print row
...
>>> for element in b.flat:
...       print element,
...
>>>
```

Iterate over rows (1st axis)

Iterate over all elements

# A simple NumPy example

- NumPy offers a large number of useful functions, see http://www.scipy.org/Numpy_Example_List
- Calculate min, max, mean, standard deviation, median of a list of numbers in a 1-dimensional array

```
>>> a = random.random ( (10) )
>>> a.min()
>>> a.max()
>>> a.mean()
>>> a.std()
>>> median(a)
```

# Read and write text file

(ex11.py)

```
>>> from numpy import *

>>> data = loadtxt("myfile.txt")  # myfile.txt contains 4 columns of numbers
>>> t,z = data[:,0], data[:,3] # data is a 2D numpy array, t is 1st col, z is 4th col

>>> t,x,y,z = loadtxt("myfile.txt", unpack=True) # to automatically unpack all columns
>>> t,z = loadtxt("myfile.txt", usecols = (0,3), unpack=True) # to select just a few columns
>>> data = loadtxt("myfile.txt", skiprows = 7) # to skip 7 rows from top of file
>>> data = loadtxt("myfile.txt", comments = '!') # use '!' as comment char instead of '#'
>>> data = loadtxt("myfile.txt", delimiter=';') # use ';' as column separator instead of whitespace
>>> data = loadtxt("myfile.txt", dtype = int) # file contains integers instead of floats
```

Matplotlib also provides similar functions called save and load

Also genfromtxt

```
>>> from numpy import *
>>> savetxt("myfile.txt", data) # data is 2D array
>>> savetxt("myfile.txt", x) # if x is 1D array then get 1 column in file.
>>> savetxt("myfile.txt", (x,y)) # x,y are 1D arrays. 2 rows in file.
>>> savetxt("myfile.txt", transpose((x,y))) # x,y are 1D arrays. 2 columns in file.
>>> savetxt("myfile.txt", transpose((x,y)), fmt='%6.3f') # use new format instead of '%.18e'
>>> savetxt("myfile.txt", data, delimiter = ';') # use ';' to separate columns instead of space
```

For binary files see http://www.scipy.org/Cookbook/InputOutput

# SciPy

- SciPy is an Open Source library of scientific tools for Python. It depends on the NumPy library, and it gathers a variety of high level science and engineering modules together as a single package. SciPy provides modules for
  - statistics
  - optimization
  - numerical integration
  - linear algebra
  - Fourier transforms
  - signal processing
  - image processing
  - ODE solvers
  - special functions
  - and more...
- See http://docs.scipy.org/doc/scipy/reference/tutorial/

# Example: Numerical integration

- Integrate a Bessel function jv(2.5,x) from 0 to 4.5: $I = \int_0^{4.5} J_{2.5}(x)\, dx.$

```
>>> import numpy as np
>>> import scipy as sp
>>> from scipy import integrate   # scipy sub-packages have to be imported separately
>>> from scipy import special
>>> result = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)
>>> print result
```

A lambda function is a small anonymous function that is restricted to a single expression – here x is the argument and the function calculates the Bessel function for value x

Limits of integration

```
>>> I = sqrt(2/pi)*(18.0/27*sqrt(2)*cos(4.5)-4.0/27*sqrt(2)*sin(4.5)+ :
...       sqrt(2*pi)*special.fresnel(3/sqrt(pi))[0])
>>> print I
```

Compare to analytical solution

$$I = \sqrt{\frac{2}{\pi}} \left( \frac{18}{27}\sqrt{2}\cos(4.5) - \frac{4}{27}\sqrt{2}\sin(4.5) + \sqrt{2\pi}\mathrm{Si}\left(\frac{3}{\sqrt{\pi}}\right) \right), \qquad \mathrm{Si}(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right) dt.$$