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

Tim O'Brien
Room 3.214 Alan Turing Building
tim.obrien@manchester.ac.uk
http://www.jb.man.ac.uk/~tob/python.html

Assessment

- Coursework : 67%
  - Two pieces of course work:
    one on Python programming – Tim,
    one on radio imaging analysis – Neal
- Exam : 33%
  - Written exam in January:
    Two compulsory questions,
    one on Python, one on radio imaging

Course Materials

- Course materials for this part will be on my website at http://www.jb.man.ac.uk/~tob/python.html and on Blackboard linked from the Student Portal at http://www.studentnet.manchester.ac.uk
- **Recommended texts**:
  - Python documentation at http://www.python.org/doc/
  - NumPy documentation at http://numpy.scipy.org/
  - SciPy documentation at http://www.scipy.org/
  - Synthesis Imaging in Radio Astronomy II, eds. Taylor, Carilli, Perley, 1999, ASP Conf. Ser. 180, (+ online material from recent Synthesis Imaging Workshops)
  - CASA documentation at http://casa.nrao.edu/

Syllabus

Part 1. Python (4-5 lecture + practicals in your own time)

Introduction to Python

Python and other languages (e.g. C/Fortran). Data types and variables. Lists, tuples and dictionaries. List data operations; list generation.

Control flow: loops and looping. Dictionary loop functions. Function definitions, packages and administration. Classes and use with arrays.

Input and output: string and file I/O.

Use of Python library functions.

Python packages

NumPy and SciPy: Python packages for scientific computing.

NumPy arrays, array indexing and array operations. Stacking and splitting, element operations. Input and output from files. String operations.

Use of some library functions: FFTs, linear algebra, random number generation/Monte Carlo methods, sorting and searching, mathematical functions.

Applications of SciPy. Input and output from fits files.

Plotting using Matplotlib. Enhancements from astronomy packages; wcsgrid, astLib.

Python

- **(Does everybody have a computer account on our astrophysics systems? Google "Basic Unix help")**
- Python is a high-level modern programming language
- You don't have to install it – it is available on our computer systems simply by typing:

  `python`

  (exit by typing ctrl-d)
- We will follow the Tutorial for Version 2.7 at http://www.python.org/doc/

An example from C

- Editing, compiling, linking and running the classic "Hello world" program in C:

```
#include <stdio.h>
main()
{
printf("hello, world\n");
return 0;
}
```

- Edit this: `emacs ex1.c`
- Compile/link with: `gcc ex1.c`
- Run with: `./a.out`

Source code

Executable (can give it a name with, for example `gcc ex1.c -o ex1`)

Same example in Fortran90

```
program ex1
write(*,*) 'hello, world'
endprogram ex1
```

- Edit this: `emacs ex1.f90`
- Compile/link with: `gfortran ex1.f90`
- Run with: `./a.out`

Same example in Python

- Python is interpreted i.e. compilation/linking happens at run-time
- The Python interpreter can either:
  - accept commands interactively from the command line;
  - or, when called with a filename as an argument, it reads and executes commands as a script from that file.

Same example in Python

- Interactive: (after typing python at the unix prompt)

```
>>> print “hello, world”
hello, world
>>>
```

- Script:
  - Edit a file e.g. `emacs ex1.py &` (ex1.py) containing:

```
print “hello, world”
```

  - Run it by typing `python ex1.py`
- Directly executable script:
  - Add a first line to your script:

```
#! /usr/bin/env python
print “hello, world”
```

  - Make the file executable with `chmod +x ex1.py`
  - Run it directly by typing `./ex1.py`

Formally, a scripting language allows you to run other software applications from within it

Note you can include comment lines in a script by starting them with #

Simple arithmetic 1

- Using the interpreter at the command line (just type in the arithmetical expressions):

```
>>> 2+2
>>> 10-3
>>> 3*6
>>> 5/2
>>> 5/-2
```

This is integer arithmetic – note in the division it returns the *floor*

- *Assignment* expressions

```
>>> width = 10
>>> length = 25
>>> length*width  # calculate area
>>> 2*_
```

Value of length*width is displayed (the # indicates the rest of the line is a comment)
Try `area=length*width`
Then type `area` to see its value.
Last printed expression is expressed as _ (makes it easy to carry on calculation)

```
>>> length = length + 10
```

Assignment not *equation*! Expression on right side of assignment is evaluated first and then assigned to variable on left hand side

Simple arithmetic 2

- Floating point operations:

```
>>> 2.2*3.5
>>> 2*4.2
>>> 3/7
>>> 3/7.0
>>> float(2)
>>> int(2.6)
```

In mixed type calculations, integers are converted to floating point.
Can convert integers to floating point and vice versa

- Complex numbers:

```
>>> 1j * 1j
>>> (1+1j)*complex(1,-1)
>>> (1+2j)/(2-1j)
>>> a = 1+2j
>>> a.real
>>> aimag = a.imag
>>> abs(a)
```

A+Bj notation used, or create with complex(A,B)

Extract real and imaginary parts from a complex number, find magnitude

Strings

- A string is a sequence of characters

```
>>> “Hello this is a string”
>>> ‘So is this’
>>> ‘I\’ve got a single quote inside me’
>>>  “I’ve got one too”
```

Single quotes or double quotes can be used, careful about mixing them.

```
>>> hello = “This is a long line of text\n\
which contains several line breaks\n\
indicated by \n and continuations to next
line indicated by \”
>>> print hello
```

Using strings or numbers directly e.g. “Hello” or 3.5 is known as a literal, otherwise assign them to variables e.g. Word=“Hello”

```
>>> shine = “All work and no play”
>>> 5*(shine + “ “)
```

Concatenate with +, or just place adjacent literals e.g. “Make” “a” “sentence”

```
>>> word = “Joist”
>>> word[0]
>>> word[1:3]
>>> word[:2]
>>> word[2:]
```

Can slice strings into substrings with index notation. Note first index is 0, can use –ve to count from the right