Introduction to IT for UCL Astrophysicists

L. Whiteway, T. Wilson, W. Jennings, E. Edmondson,
L. Lucie-Smith

Department of Physics and Astronomy
University College London

13 & 20 October 2016

Where to find this presentation

`http://tinyurl/hhndmla`

Overall goals of presentation

- What software you might find useful
- Where to get more information (UCL courses, web, etc.)
- UCL-specific information (e.g. login details)
- Some hands-on work

Specific contents

13 October:

- Accessing Astrophysics group machines
- Using the Linux console
- Basics of Python

20 October:

- Commonly used programs (LaTeX, DS9, IRAF,...)
- Using High-Performance Computing (HPC) machines
- HPC best practices

A further session on 27 October will cover machine learning using Python.

Information on the Web

Astrophysics Wiki

`https://wiki.ucl.ac.uk/display/PhysAstAstPhysGrp/Main+Page`

This Wiki is freely viewable and editable by all members of the department. Please use it to record information that you think will be useful to others (including your future self). Be bold!

UCL Research Computing Platforms

`https://wiki.rc.ucl.ac.uk/wiki/Main_Page`

Stack Overflow

`http://stackoverflow.com/`

Computing Environment for Astrophysics

- Large datasets requiring substantial processing followed by sophisticated statistical analysis
- Calculations often done on specialised 'high-performance computing' (HPC) machines having large filesystems and large RAM; calculations are often broken into pieces that can be run simultaneously ('in parallel') across many processors.
- Much useful software is made freely available within the community. Software quality is usually high; documentation quality is more variable.
- Many users write their own software.

Local Computing Environment

You will have your own local machine, which might be:

- PC (Windows)
- Mac
- Linux

Also there are shared Linux machines:

- General purpose Astrophysics server available from outside UCL: `zuserver1`
- UCL Cosmology HPC cluster: `splinter`
- Other UCL HPC clusters: `Grace`, `Legion` and `Emerald`
- National HPC cluster: `DiRAC`

Work patterns

Several work patterns are possible:

- Write and test a program on your local machine; use the local machine to remotely connect to `splinter`; upload the program to `splinter` and run it there;
- Or do all your work locally (requires small data sets);
- Or use the local machine to remotely connect to `splinter` and do all your work there.

Remote connections

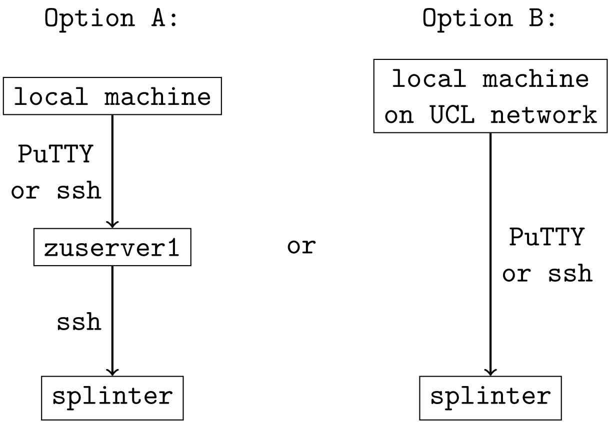
How to connect to a shared machine

- Windows PC: use PuTTY;
- Mac: go to the Terminal window and use `ssh`;
- Linux machine: go to the Terminal window and use `ssh`.

Visibility

If you are not on the UCL network then you cannot connect to `splinter` directly; instead you must go via `zuserver1`.

Two methods for accessing splinter



Accessing remote machines

Credentials

- You will need a *username* and *password* for `splinter` (and, if you are using Option A, for `zuserver1` as well).
- If you do not have these already then we can give guest credentials to be used during this course.

The full names of the servers are:

- `zuserver1.star.ucl.ac.uk`
- `splinter-login.star.ucl.ac.uk`

Using PuTTY for remote connections from Windows

- If you don't have PuTTY you can download it from `http://www.putty.org/`.
- On the 'Connection/SSH/X11' tab, click on 'enable X11 forwarding' and set 'X display location' to 'localhost:0' - this is necessary for handling graphical output.
- On the Session tab, set the Host Name as appropriate: `zuserver1.star.ucl.ac.uk` (Option A) or `splinter-login.star.ucl.ac.uk` (Option B).

Using ssh for remote connections from Mac and Linux

- Syntax: `ssh -YC username@servername`
- The ‘Y’ option is necessary for handling graphical output.

X-Windows client

- If the remote program that you are running produces graphical output, then you must have a program (an 'X-Windows client') running on your local machine to display this graphical output.
- On Windows you can use XMing (`https://sourceforge.net/projects/xming/`) or Exceed (available on the UCL Desktop).
- On Mac you can use XQuartz.
- On Linux you can sit back and relax as the graphical interface is an X-server.

Linux: Command shell

- In Linux you will use a ‘command shell’.
- This is a text-based environment in which you type commands and receive text output.
- Not GUI! Reflects the hardware limitations current when Unix was created. Low-tech and reliable e.g. for remote access.
- Various command shell programs are used: `bash`, `csh`, `tcsh`, etc. To see which one you are using, call `echo $0`.

Linux: Directory structure

- Everything is organised around files (which may be data files or program files i.e. instructions to be executed).
- Files live in directories. There is a hierarchical tree structure of directories.
- Sample file name:
  `/share/splinter/ucapwhi/des/foo.txt`
- Note use of slash '/', not backslash '\' as in Windows.
- Case sensitivity: 'Foo' and 'foo' are different strings.

Linux: Special symbols for directories

Symbol	Meaning
/	Top of the directory tree (the root directory)
.	Current directory
..	Parent of the current directory
~	User's 'home' directory

Linux: Environment variables

- The operating system maintains a global namespace of ‘environment variables’ to store configuration information.
- Use `printenv` (in tcsh) or `set` (in bash) to see all environment variables; use `echo $<variable_name>` to see the value of one environment variable (e.g. `echo $PATH`).
- Use `setenv FOO my_string` (in tcsh) or `export $FOO='my_string'` (in bash) to set `FOO`.
- Variables `PATH` and `PYTHONPATH` are used frequently (to maintain lists of directories in which to search for executable programs and Python modules, respectively).
- Linux has no equivalent of the Windows Registry; configuration is done via the directory structure and the environment variables.

Linux: Structure of commands

Structure

```
[command] -[option(s)] [argument]
```

Examples

```
ls -la
mkdir hello_world
cp hello.cpp new_hello.cpp
```

Linux: command reference

There is a very useful summary of Linux commands at:
`http://www.computerhope.com/unix.htm`

Linux: Basic commands 1

navigation and help

```
pwd
ls -la
cd dir_name
man command_name
info command_name
exit
```

copy or move

```
cp src dest
mv src dest
scp usr@host:file dest
```

create or delete

```
touch file.txt
mkdir dir_name
rm -i file.txt
```

find and system info

```
whereis file
which
echo $VAR_NAME
```

file contents

```
cat file
more file
head file
```

Linux: Basic commands 2

special characters

& (background)
; (combine)
\ (next line)
* (wildcard)
| (pipe)
> (output)
< (input)

text editors

emacs
vi
gedit

process management

kill, top, nohup

compressed files

gunzip, tar

images

gthumb
ds9
evince
eog

publishing

latex, bibtex

Linux: Exercises I

1. Go to your home directory and create a directory called `level_0`.
2. Change directory to `level_0`.
3. Find the name of the present working directory.
4. Make a directory `level_1`, and move to it.
5. Create a file called `foo.txt` with contents “This file contains the word bar”.
6. Add another line in `foo.txt` with contents “This is the second line”.
7. Print the contents of `foo.txt` to the screen.
8. Search for the word *bar* in `foo.txt`.
9. Go up one level, then remove the directory `level_1` (and its contents).
10. Find the location of your python installation.

Linux: Exercises II

11. Find the values of the environment variables `PATH` and `LD_LIBRARY_PATH`.
12. Set the environment variable `MY_VAR` to equal the absolute path to `level_0`, and test that it has worked OK.
13. Add (i.e append) to the `PATH` the absolute path to `level_0`, and test that it has worked OK.
14. Use the `man` command to find the option of `ls` that shows file sizes in human readable format.
15. Find the hostname, processor type and operating system version and write this info into a text file called `info.txt`.
16. List the people who are currently logged into the system.
17. Find which process is using the most CPU at the moment.
18. Find the IDs of the processes that you are running.

Programming: Languages

High-level languages - fast to code

- Python
- IDL
- Matlib, Mathematica, R

Low-level languages - fast to run

- C, C++
- Fortran
- Assembler

Python

Python has become popular:

- Good trade-off between ease-of-use and performance
- Many add-in libraries, so sophisticated programs can often be built easily [`https://xkcd.com/353/`].

Python Features and Aspects

- Two slightly-incompatible versions: 2 and 3 (currently 2.7 and 3.5). We tend to use 2.7.
- Can be used interactively or compiled.
- Duck typing (type of variable is inferred at run-time)...
- ... and hence generic programming (in which a function can take many different types of variables as inputs).
- Object-orientation.
- Exceptions (try, raise, except) for handling error conditions.
- Automatic garbage collection (so no need to worry about memory management).

Python for Astrophysicists

Use Python plus the following add-in libraries:

- Numpy: array processing for numbers and strings
- Scipy: scientific library
- Astropy: astronomical library
- Matplotlib: plotting

All of these libraries (plus many others) are available in one package called Anaconda:

`https://www.continuum.io/downloads`.

Using Python on `splinter`

- To initialize your `splinter` session so that it will use Python 2.7 and so that it can find all the necessary libraries, run: `module load dev_tools/oct2015/python-Anaconda-2-4.0.0`
- This will set the necessary environment variables.
- We will explain in more detail later about the `module` command.

Python: Interactive

- Begin an interactive Python session by calling `ipython`
- You can then type Python commands on successive lines, such as

```
print 2+2
a = 7
print a**2
exit
```

Python: Compiled

- Create a Python program by writing code in a text file (say called my_program.py).
- Then execute the program by calling `python my_program.py`
- Even better:
  - Put `#!/usr/bin/env python` as the first line in my_program.py
  - Run `chmod +x my_program.py`

  Then execute the program by calling `./my_program.py`

Python: sample program

```
#!/usr/bin/env python
# Use the '#' to indicate a comment

# Import library and give it a
# short name for convenience
import numpy as np


a = np.arange(10) # The array 0, 1, ..., 9
b = f(a) # f is a subroutine defined below
print b

def f(x):
    # The whitespace at the beginning
    # of the next line is crucial.
    return x**2
```

Python

Dictionaries

Useful to do key-value mapping and can be created using `dict()`
Can access keys using `dictionary.keys()` and values using `dictionary.values()` or `dictionary[key]`

Functions

```
def function_name(arg1, arg_vol=1, *args, **kwargs):
```

`arg_vol=` is a voluntary argument, and if you don't want to specify the number of arguments use `*args` for a list of arguments and `**kwargs` for a dictionary, for example:

```
function_name(''HelloWorld'', *[1, 2, 3],
    **{''foo'':''bar''})
```

Classes

Can create objects with functions, but must initialise the arguments:

```
class class_name:
    def __init__(self, arg):
        self.argument = arg
```

NumPy

NumPy Arrays

Can create NumPy arrays in many ways; `np.array([])`, `np.empty(())`, `np.zeros([])`, `np.ones([])`, `np.arange()`, `np.linspace()`, ...

Basic Statistics

`np.mean()`, `np.median()`, `np.min()`, `np.max()`, `np.std()`, `np.argmin()`, `np.argmax()`

Shape Manipulation

Can change arrays to be different shapes and size; `np.reshape()`, `np.flatten()`, `np.shape()`, array slicing (next slide...)

Sorting

Can use `np.sort()` to sort arrays along different axes, and `np.argsort()` to return the arguments of the sorted arrays

I/O

Can use `np.loadtxt()` and `np.genfromtxt()` to get values from a data files. Can choose data type, delimiter, to skip header/footer/rows, to unpack the data into multiple variables, etc.

Array Slicing

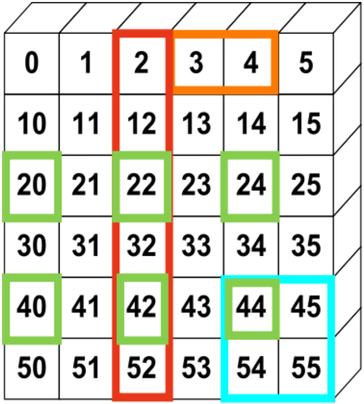
```
a = np.arange(60).reshape(6,10)[:,0:6]
```

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,22,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



Scipy/Astropy/PyFITS/SymPy

Constants and Conversions

For example: `scipy.constants.c`, `scipy.constants.hbar`, `scipy.constants.lambda2nu(550*scipy.constants.nano)`

More Stats

Various stats tool using `scipy.stats`, such as calculating mean, variance, skew, and probability and cumulative density functions

Optimising and Fitting

Can use `scipy.optimize` to find minima (`.fmin_bfgs`), find roots of a function (`.fsolve`), and fitting (`.curve_fit`)

PyFits

Using `pyfits`, can import (`.open()`), read (`.info()`/`.header()`), and write (`.PrimaryHDU()` and `.writeto()`) FITS files

World Coordinate Systems

Can find the RA and Dec of pixels using `astropy.wcs.WCS` and and convert RA and Dec to pixels using `astropy.wcs.WCS.wcs_sky2pix()`

Symbolic Calculus

Differentiation, Integration, Linear Algebra, Series Expansion, and Equation Simplifying using `sympy`

Information on the Web

Documentation

http://scipy.org/
http://matplotlib.org/
http://www.astropy.org/

SciPy Tutorials (Also NumPy and Matplotlib)

https://conference.scipy.org/scipy2013/tutorials.php

SciPy Lectures (Also NumPy and Matplotlib)

http://www.scipy-lectures.org/

Stanford's Introduction to Scientific Python

http://web.stanford.edu/~arbenson/cme193.html

Matplotlib

- Powerful plotting library. See `http://matplotlib.org/gallery.html` for range of examples.
- Two interfaces: one similar to MATLAB (for interactive use; relies on global state), and one object-oriented. Typically the latter is to be preferred.
- Read the introduction at `http://matplotlib.org/faq/usage_faq.html` as this clarifies many points of terminology and usage.

Matplotlib example

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

x = np.arange(0, 2*np.pi, 0.1)
y = np.sin(x)

plt.figure()
plt.scatter(x, y) # Or plt.plot...
plt.show()
```

Matplotlib Features

Different Plotting Styles

Can edit marker style, colour, edge colour, size and opaqueness, errorbar style and colour, can annotate plots with text and arrows, and can include colour maps and bars.
Can plot in polar and World Coordinate System coordinates (with the help of `astropy`).

Image Plotting

Can read and plot an image using `matplotlib.cbook` and `plt.imshow`

Multiple Subplots

Using `plt.subplot()` you can create multiple subplots, for example the following creates two side-by-side subplots that share x and y axes:

```
plt.subplots_adjust(hspace=0.5)
ax1 = plt.subplot(121)
ax2 = plt.subplot(122, sharex=ax1, sharey=ax1)
```

Matplotlib exercise

- Use the SDSS data file (compressed FITS format) `/share/splinter/ucapwhi/Linux_training/demo.fits.gz`.
- Use the pyfits library (`import astropy.io.fits as pyfits`) to manipulate file columns as numpy arrays.
- Use the `open` command to get a handle to the file (`h = pyfits.open(...)`). Then get e.g. the `Foo` column via `x = h[1].data.field['Foo']`.
- Create a scatter plot of r-band absolute magnitude (`'rMAG'`) versus distance (`'Dist'`). Do you think that this data set is a subset of a larger data set?
- Make a map by scatter-plotting declination (`'DEdeg'`) versus right ascension (`'RAdeg'`). Use small dots. Why is the density lower at the top? Why should the plot really be flipped left/right?

Source control system

- A source control system is a repository for successive versions of documents (e.g. source code for computer programs).
- Versions can be compared and old versions restored if needed (e.g. to undo recent bad changes).
- Multiple developers are able to make changes to the same file, with automatic merging of edits from different developers (but if there is an *edit conflict* i.e. two developers change the same line then manual intervention will be needed).
- Any serious development that you do (even if just for yourself) should be under the control of a source control system.

Git

TODO: Describe Git, draw Git diagram

GitHub and BitBucket

TODO: Describe Github and Bitbucket

Common and Useful Programs

LaTeX
Talk about Tikz in here

DS9

IRAF

SExtractor/SWarp

DS9

Features

- Aligning with WCS (World Coordinate System)
- Scaling image contrast
- Funky colour sets (and inverse or negative)
- Regions and annotating the image
- Multiple frames; blinking and matching
- Contour plots

DS9 Website

`http://ds9.si.edu/site/Home.html`

Image Reduction and Analysis Facility (IRAF)

Features

- Uses basic Linux commands to navigate around directories, copy/move files, run tasks in the background
- A series of packages each containing various tasks
- `?` to view all tasks in current package, `??` to view all tasks
- Each tasks contains various editable parameter files
- `lpar [parameter_file_name]` to view and `epar[parameter_file_name]` to edit a parameter file
- `unlearn [parameter_file_name]` to set reset task parameters

IRAF Websites

http://iraf.noao.edu/

http://iraf.net/irafdocs/

SExtractor/SWarp

Features

- *"SExtractor is a program that builds a catalogue of objects from an astronomical image. Although it is particularly oriented towards reduction of large scale galaxy-survey data, it can perform reasonably well on moderately crowded star fields."*
- *"SWarp is a program that resamples and co-adds together FITS images using any arbitrary astrometric projection defined in the WCS standard."*

SExtractor & SWarp Websites

http://www.astromatic.net/software/sextractor
http://www.astromatic.net/software/swarp

CosmoSIS

HPC: Information on the Web

This presentation

https://github.com/Astrophysics-UCL/HPCInfo/

Splinter on the UCL Astrophysics Wiki

https://wiki.ucl.ac.uk/display/PhysAstAstPhysGrp/Splinter+User+Guide

UCL Research Computing Platforms

https://wiki.rc.ucl.ac.uk/wiki/Main_Page

DiRAC

http://www.dirac.ac.uk/

HPC: Mailing list

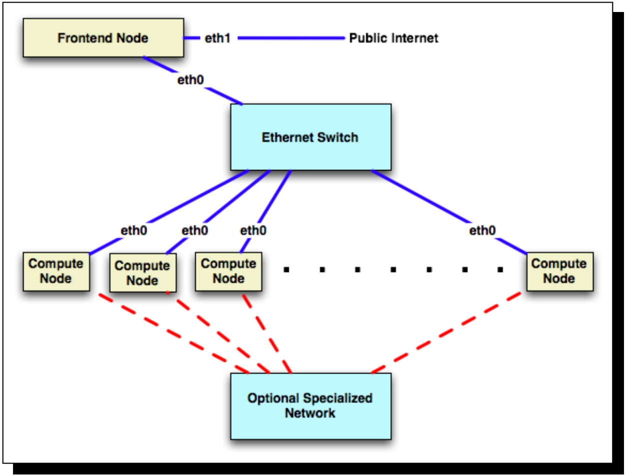
`https://www.mailinglists.ucl.ac.uk/mailman/listinfo/splinter-users`

- please subscribe
- post any issues regarding splinter

HPC: Splinter specs

- As of October 12, 2016, *splinter* has 528, 4TB memory
- 8 nodes, dual 6-core 2.8GHz, 48GB memory
- 20 nodes, dual 8-core 2.0GHz, 128GB memory
- SMP node, 40 2.4GHz cores, 1TB memory
- login node, dual 10-core, 2.4GHz 98GB memory
- head-node, dual 8-core, 2.4GHz, 164GB memory

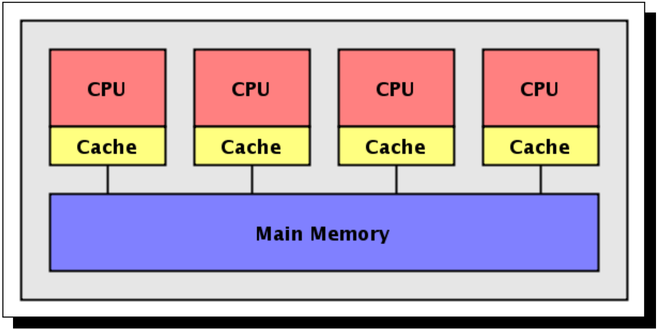
HPC: splinter distributed



[1]

[1] http://www.rocksclusters.org/

HPC: splinter shared



[2]

[2] http://www.cs.rit.edu/

HPC: Workspaces I

`/home/user_name`

- this is your home directory
- login scripts can be put here
- 1GB quota
- private

`/share/splinter/user_name`

- create the directory if not already there
- can be used as a workspace
- no quota
- public unless made private

HPC: Workspaces II

`/share/data1`

- for storing large data
- you can create a directory for your, .e.g, `/share/data1/SKA`

`/share/apps`

- for installing software
- module-files

HPC: Login script

- everytime you login this file will be executed
- this file is in your `$HOME`
- it is called `.login`
- you can load modules, envvars, etc.

Examples

Load my aliases

```
source ~/aliases.csh
```

Load python

```
module load dev_tools/nov2014/python-anaconda
```

HPC: Modules

- easy and flexible way use software
- available to everyone in splinter

Examples

Print the available modules

`module avail`

Load a module

`module load module_name`

List the loaded modules

`module list`

Unload a module

`module unload module_name`

Unload all modules

`module purge`

Help

`module --help`

HPC: Submitting jobs

- computing jobs should be submitted to the scheduler
- you will have to write a job script
- interactive job

Examples

Submit a job

```
qsub job_script
```

Submit an interactive job

```
qsub -I
```

Check the status of a job

```
checkjob job_id
```

List the status of all jobs

```
qstat
```

Show the queue

```
showq
```

Delete a job

```
qdel job_id
```

HPC: Queues

- `compute`
- `cores16`
- `cores12`
- `smp`

HPC: Structure of a job script

```
#!/bin/tcsh
# PBS -q cores12
# PBS -N a_name_for_your_job
# PBS -l nodes=1:ppn=6
# PBS -l mem=32gb
# PBS -l walltime=120:00:00
```

Set some environment variable

```
setenv OMP_NUM_THREADS 6
```

Source paths if needed

```
source /home/username/libpaths.csh
```

Run my program

```
/home/username/hello_world.exe
```

HPC: Jobscripts: things to remember

- Submit the job to the right queue
- Request the correct number of `nodes` and `ppn`
- Specify the memory required
- Always specify the walltime
- If your program is not parallel, please use `nodes=1,ppn=1`
- Use `-q compute` for single processor jobs
- Use `qsub -I` for interactive job
- If using most of the resources, please send an email to the mailing list.

HPC: More PBS commands

Specify output

```
PBS -o path/to/file.out
```

Specify error output

```
PBS -e path/to/file.err
```

Mail alert at (b)eginning, (e)nd, and (a)bortion of execution

```
PBS -m bea
```

Send mail to the following address

```
PBS -M your_email_id@ucl.ac.uk
```

HPC: Using *Ganglia*

`http://splinter.star.ucl.ac.uk/ganglia/`

- is tool for analysing splinter
- can only be loaded from splinter (using firefox)
- will give you load/memory information
- can look into nodes

HPC: Collaborative projects

- collaboration between two splinter users
- can share common data in `/share/data1/my_collaboration`
- give read/write permission to other users using `chmod`

HPC: Best practices

- Choose the machines that are suited for your problem
- Read the User Guide
- Do not run your programs in the login node
- Install common software locally if and only if absolutely necessary
- Request optimum resources
- Minimise data transfer between nodes,
- Backup! Backup! Backup!

HPC: Exercises

https://github.com/Astrophysics-UCL/HPCInfo/tree/master/training/workshop_2016/

Extra Slides - Python Examples

Dictionary example:

```
dc = dict(a=1, b=''Hello World'', c=(1, 2, 3))
print dc.keys(), dc.values(), dc[''b'']
```

Function example:

```
def my_first_func(arg1, arg_vol=1, *args, **kwargs):
    print arg1, arg_vol
    for a in args:
        print a
    for key, value in kwargs.iteritems():
        print key, value
```

Class example:

```
class my_first_class:
    def __init__(self, arg):
        self.argument = arg

    def print_arg(self):
        return self.argument

instance = my_first_class(42)
print instance.method()
```

Extra Slides - NumPy Examples

Shape Manipulation examples:

```
a = numpy.arange(60).reshape(6,10)
print a.shape
print a.flatten()
print a.reshape(2, -1, 3)
```

Sorting examples:

```
a = numpy.array([2, 6, 5, 1, 6, 3, 3])
print numpy.sort(a)
print numpy.argsort(a)

data_type = [(''wavelength'', int), (''flux'', float)]
values = [(31, 210), (45, 3400), (18, 150), (7, 50), (21, 100)]
array = numpy.array(values, dtype=data_type)
print numpy.sort(array, order=''wavelength'')
```

I/O examples:

```
data = numpy.genfromtxt(file_name, dtype=data_type,
    delimiter=delimiter_character, skip_header=number_of_lines_to_skip)
data1, data2 = np.loadtxt(file_name, skiprows=number_of_lines_to_skip,
    unpack=True)
```

Extra Slides - Scipy/Astropy/PyFITS/SymPy Examples 1

Constants and conversions example:

```
x = scipy.constants.find('Newton')
print scipy.constants.value(x[0]), scipy.constants.unit(x[0])
```

More stats examples:

```
bell = scipy.stats.norm(loc=centre, scale=standard_deviation)
print bell.stats(moments='mvs')
print bell.pdf([value, value_2, value_3])
print bell.cdf([value, value_2, value_3])
```

Optimising and Fitting examples:

```
minima1 = scipy.optimize.fmin_bfgs(function, initial_guess)
minima2 = scipy.optimize.brute(function, (search_grid,))

r1 = scipy.optimize.fsolve(function, initial_guess)
r2 = scipy.optimize.newton_krylov(function, initial_guess, verbose=True)

opt_values, covariance_matrix = scipy.optimise.curve_fit(function, x_data, y_data)
plt.plot(x_data, y_data, 'bo', label='data')
plt.plot(x_data, function(x_data, opt_values[0], opt_values[1]), 'r-', label='fit')
```

Extra Slides - Scipy/Astropy/PyFITS/SymPy Examples 2

PyFITS examples:

```
fh = pyfits.open(file_name)
print fh.info(), fh[1].header()

hdu = pyfits.PrimaryHDU(data)
hdu.writeto(new_image_name, clobber=True)
```

World Coordinate System examples:

```
wcs = astropy.wcs.WCS(header=pyfits.open(file_name)[1].header)
pixcrd = numpy.array([[coord1_x, coord1_y], [coord2_x, coord2_y]], numpy.float_)
sky = wcs.wcs_pix2sky(pixcrd, 1)
print sky

pixcrd2 = wcs.wcs_sky2pix(sky, 1)
print pixcrd2
```

Symbolic Calculus examples:

```
a = sympy.Symbol(''a'')
b = sympy.Symbol(''b'')
e = (a + 2*b)**5
e.diff(b)
sympy.integrate(a**2 * sympy.cos(a), a)
sympy.simplify((a + a*b)/a)
sympy.series(sympy.cos(a), a)
```

Extra Slides - Matplotlib Examples

Annotate examples:

http://matplotlib.org/examples/pylab_examples/annotation_demo2.html

Polar and color/size/opaqueness example:

```
plt.figure(figsize=(10,10))
ax = plt.subplot(111, polar=True)
c = scatter(angles, radii, c=colors, s=sizes, alpha=opaqueness)
plt.show()
```

Image plot example:

```
datafile = matplotlib.cbook.get_sample_data(file_name, asfileobj=True)
A = fromstring(datafile.read(), uint16).astype(float)
A *= 1.0/max(A)
A.shape = 512, 512
im = plt.imshow(A, cmap=cm.hot, origin='upper', extent=plot_size)
plt.show()
```

Subplot and colour bar example:

```
fig, axes = plt.subplots(nrows=2, ncols=2)
for dat, ax in zip(data, axes.flat):
    im = ax.imshow(dat, vmin=colour_min, vmax=colour_max)
cax = fig.add_axes([left, bottom, width, height])
fig.colorbar(im, cax=cax)
plt.show()
```