7. POWER IN PACKAGES

JHU Physics & Astronomy
Python Workshop 2017

Lecturer: Mubdi Rahman

NOW, FOR SOME FUN!

We've been showing you how to do things in Python that you could (for the most part) in many other scripting/programming languages. Let's show you things that make Python great!

NOW, FOR SOME FUN!

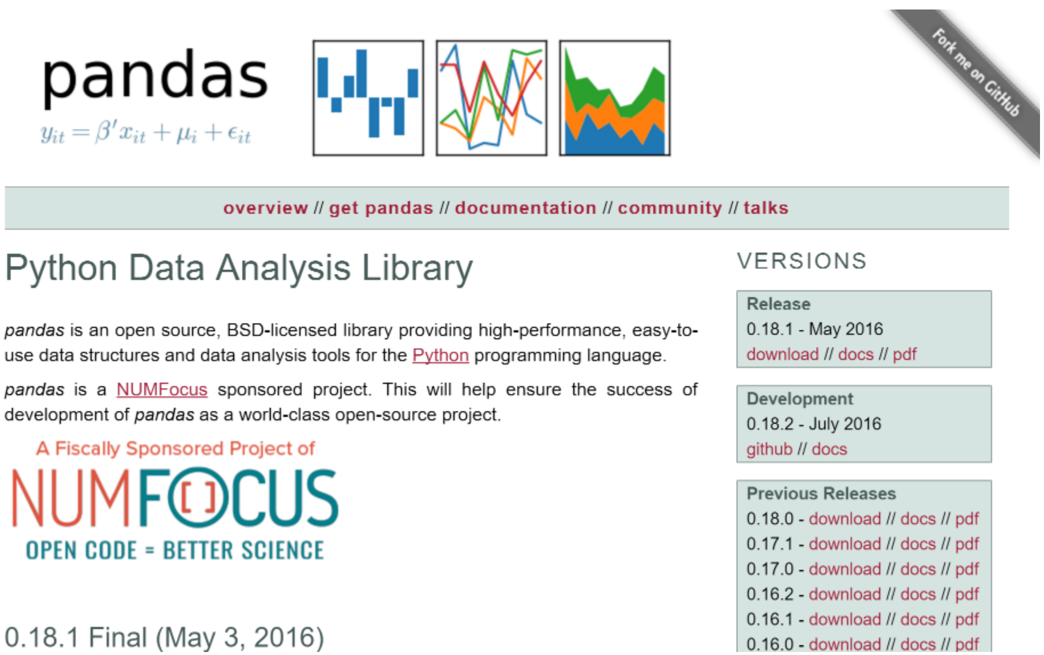
We've been showing you how to do things in Python that you could (for the most part) in many other scripting/programming languages. Let's show you things that make Python great!

PRO TIP:

In this section, we'll also be using the astroquery package. You should install this now using pip or the package manager.

THE PANDAS IN THE ROOM

Pandas is a fully-featured data analysis library. This is *well* beyond the scope of our workshop, but it allows a great deal of data mangling and manipulation features. Particularly useful for pivot tables & timeseries.



http://pandas.pydata.org

A QUICK PANDAS EXAMPLE

Quickly turning a two-dimensional dataset into a three-dimensional dataset using a column to group on:

```
import pandas as pd

# Creating Random Data:
tmp_dates = pd.date_range(‘2015-10-29’,
                          periods= 100) # Date
tmp_cat = n.random.randint(1, 4, 100) # Category
tmp_data = n.random.randn(100) # Data

# Creating Pandas Dataframe:
df1 = pd.DataFrame(
 {‘dates':tmp_dates, 'cat':tmp_cat, 'data':tmp_data}
)
```

A QUICK PANDAS EXAMPLE

```
# Now to group by category as well as date:
pdf1 = df1.pivot('dates', 'cat', 'data')

print(pdf1)

# Returns:
# cat                1         2         3
# dates
# 2015-10-29  0.515307       NaN       NaN
# 2015-10-30       NaN  0.163088       NaN
# 2015-10-31  0.972008       NaN       NaN
# 2015-11-01       NaN -0.502585       NaN
# 2015-11-02       NaN  0.274932       NaN
# 2015-11-03       NaN  0.258800       NaN
# 2015-11-04 -1.579318       NaN       NaN
```

WORLD COORDINATE SYSTEM

Typically, you need to know *where* on an image each pixel is in astronomical coordinates (either RA & Dec, or maybe Galactic Longitude and Latitude). This information (typically referred to as WCS) is typically stored in the header of your FITS file. To use this information, you can use the astropy.wcs module:

```
from astropy.wcs import WCS

# If you have a header object named ‘head1’ from
# either fits.getheader() or fits.open():
w = WCS(head1)

# Or just getting one from a file itself:
w = WCS(filename.fits)
```

FROM PIXELS TO COORDINATES

The wcs object contains functions that conversion from pixel to world coordinates and vice versa:

```
# From pixel => world:
ra, dec = w.all_pix2world(xpx, ypx, 0)# Can be lists

# The third parameter indicates if you're starting
# from 0 (Python-standard) or 1 (FITS-standard)

# From world => pixel:
xpx, ypx = w.all_world2pix(ra, dec, 0)
```

FROM PIXELS TO COORDINATES

The wcs object contains functions that conversion from pixel to world coordinates and vice versa:

```
# From pixel => world:
ra, dec = w.all_pix2world(xpx, ypx, 0)# Can be lists

# The third parameter indicates if you're starting
# from 0 (Python-standard) or 1

# From world => pixel:
xpx, ypx = w.all_world2pix(ra,
```

PRO TIP:

Note that the order of the input are in standard Cartesian ordering, and the **opposite** of the FITS image read in.

PLOTTING A FITS IMAGE

It is important to note that most often, the pixels from the FITS image are not perfectly aligned with the coordinate grid, and aren't necessarily the same size on sky throughout the image. **In these cases, it is critical to use pcolor (or pcolormesh) to get the orientations correct.**

If you want to use imshow, remember anything else you'd like to plot should be converted into pixel coordinates through the **w.all_world2pix()** function.

Next, we'll run through plotting an image:

PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, imfile[0].data
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist, 0)

# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, imfile[0].data
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, yp
ralist, declist = w.all_pix2world(

# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

Getting all of the image and header data

PLOTTING A FITS IMAGE

Making list of bin edges. Remember there are N+1 bins. The coordinates are defined on the centre of the pixel, so the first bin edge is at -0.5.

```
# Getting Data
imfile = fits.open(filename
header, im = imfile[0].head
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist, 0)

# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(f
header, im = imfile[
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist, 0)

# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

np.meshgrid() creates two, 2-D arrays filled with the values in the two 1-D arrays you give it:

```
np.meshgrid([1,2],[3,4]) =
    [1, 2]  [3, 3]
    [1, 2], [4, 4]
```

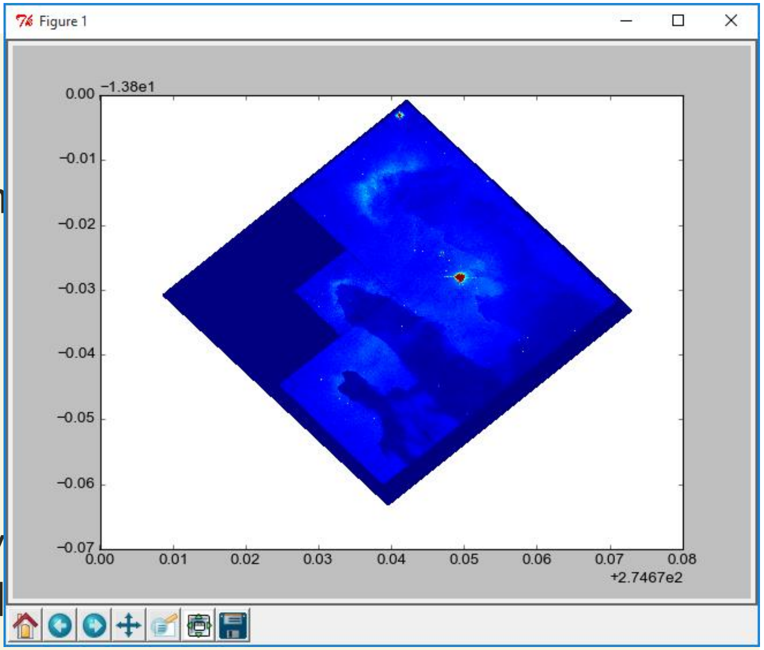
PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header,
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist, 0)

# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

Converting the indices into RA, Dec values for all values in the lists.

PLOTTING A FITS IMAGE

Final Plot

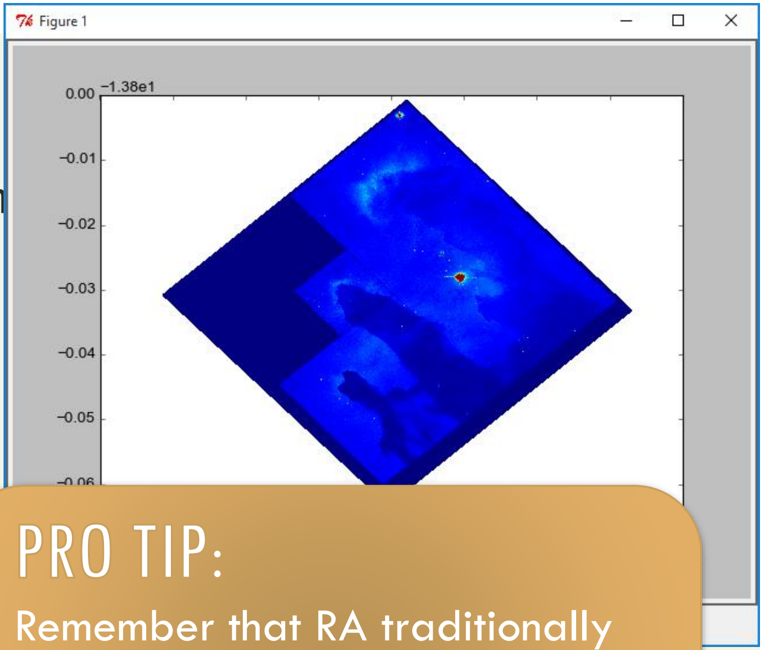



```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, im
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.
ypx = np.arange(im.shape[0]+1)-0.
xlist, ylist = np.meshgrid(xpx, y
ralist, declist = w.all_pix2world

# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, im
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.
ypx = np.arange(im.shape[0]+1)-0
xlist, ylist = np.meshgrid(xpx,
ralist, declist = w.all_pix2wor

# Plotting
plt.pcolormesh(ralist, declist,
```

Final Plot



PRO TIP:

Remember that RA traditionally increases to the left, so you'll have to flip the axis manually through plt.xlim()

COORDINATE TRANSFORMATIONS

Astropy provides a way of dealing with coordinates, and automatically deal with conversions:

```
from astropy.coordinates import SkyCoord

# Making Coordinates:
c1 = SkyCoord(ra, dec, frame='icrs', unit='deg')
c2 = SkyCoord(l, b, frame='galactic', unit='deg')
c3 = SkyCoord('00h12m30s', '+42d12m00s')

# Printing and Conversions:
c1.ra, c1.dec, c1.ra.hour, c2.ra.hms, c3.dec.dms
c2.fk5, c1.galactic # Converting Coordinates
c2.to_string('decimal'), c1.to_string('hmsdms')
```

PHYSICAL UNITS

Astropy provides a way to manipulate quantities, automatically taking care of unit conversions automatically:

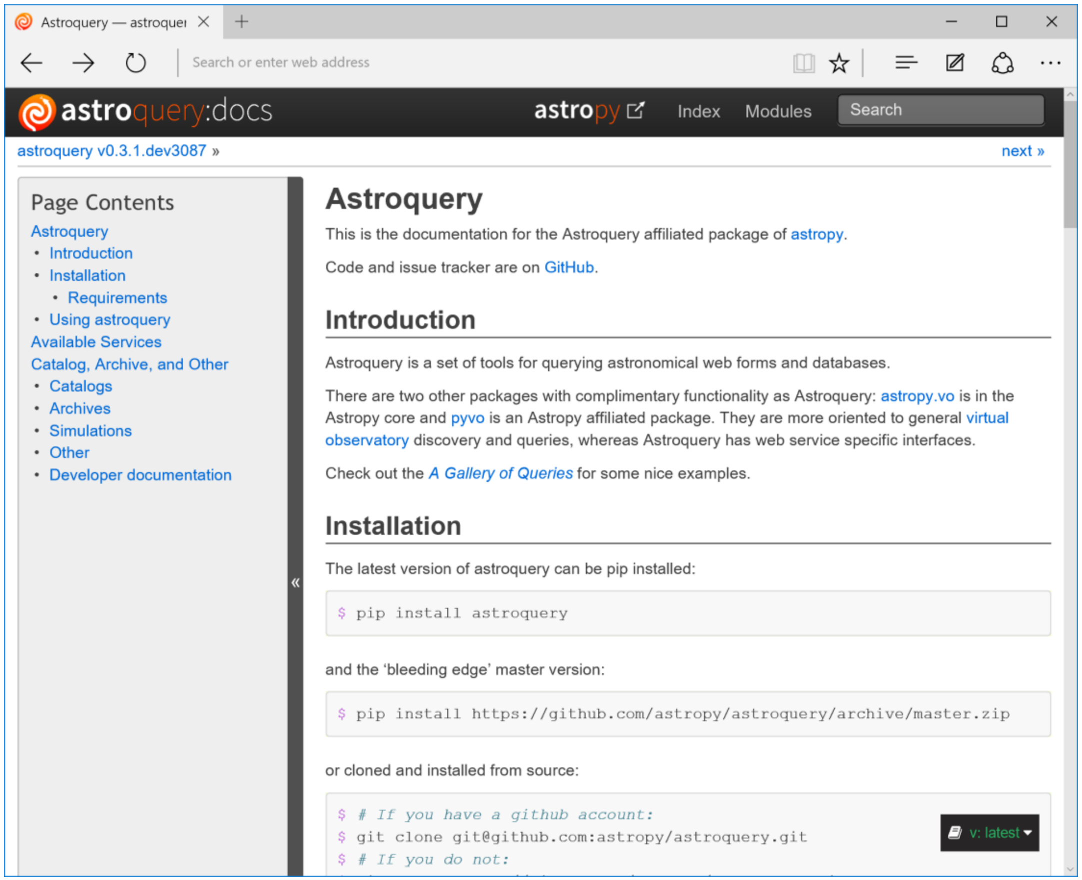
```
from astropy import units as u

# Defining Quantities with units:
val1, val2 = 30.2 * u.cm, 2.2E4 * u.s
val3 = val1/val2 # Will be units cm / s

# Converting Units
val3km = val3.to(u.km/u.s)

# Simplifying Units
val4 = (10.3 * u.s / (3 * u.Hz)).decompose()
```

PHYSICAL/ASTRONOMICAL CONSTANTS

Astropy also provides constants (with units):

```
from astropy import constants as c

# Some constants
c.k_B, c.c, c.M_sun, c.L_sun

# Can use with units
energy = c.h * 30 * u.Ghz

# Can convert units
mass =  (3.2E13 * u.kg).to(c.M_sun)
```

ASTRONOMICAL QUERYING



Astroquery allows access to online databases of various sources.

The documentation is located:

http://astroquery.readthedocs.org/en/latest/

ASTRONOMICAL QUERYING

There are lots of possible databases to query, but as a quick example (from Simbad):

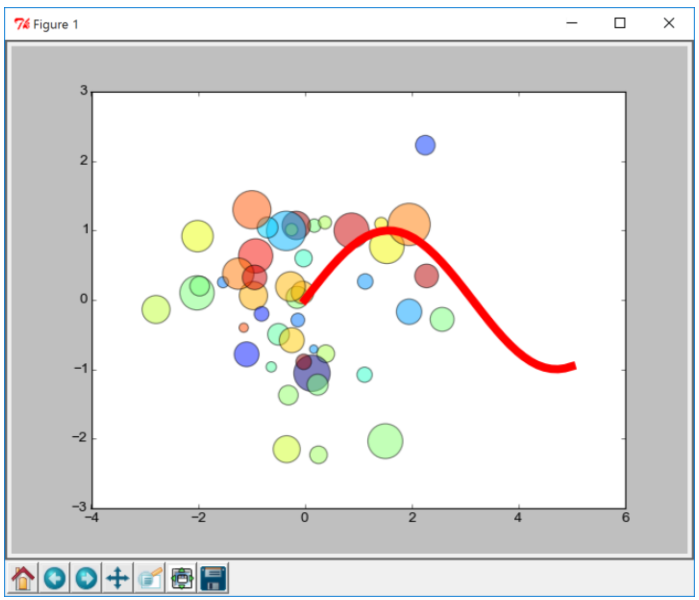
```
from astroquery.simbad import Simbad

# Simbad
s = Simbad()

# Table of Matching Objects
tab1 = s.query_object('M31')

# Printing Table
tab1.pprint()
```

ASTRONOMICAL QUERYING

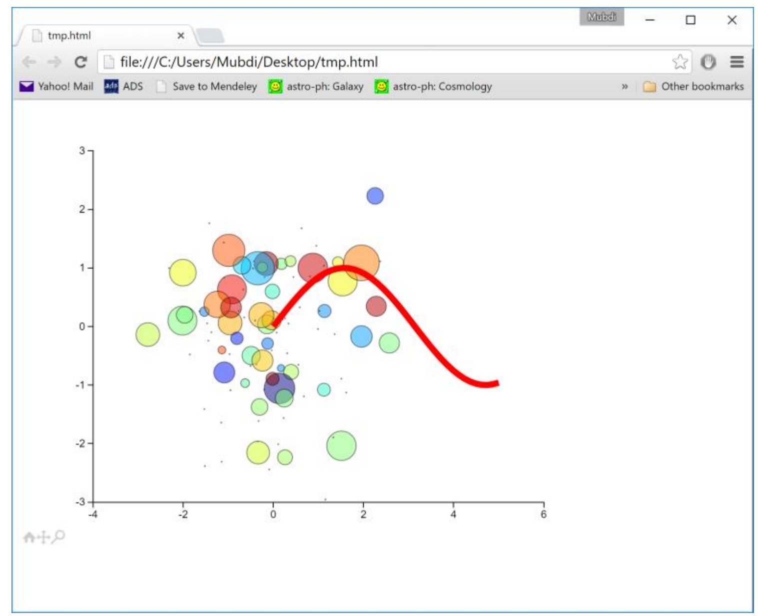
There are lots of possible databases to query, but as a quick example (from Simbad):

```
from astroquery.simbad import Simbad

# Simbad
s = Simbad()

# Searching on Region
c1 = SkyCoord(298.4, -0.4, frame='galactic',
  unit='deg')
tab1 = s.query_region(c1, radius=1*u.deg)

# Printing Table
tab1.pprint()
```

MPLD3: MATPLOTLIB IN YOUR BROWSER



Matplotlib Figure



D3.js Webpage

MPLD3: MATPLOTLIB IN YOUR BROWSER

While not every matplotlib function is supported, it is easy to export your plot into an interactive HTML-based plot:

```
import mpld3

# If you have a figure already defined: fig1
mpld3.save_html(fig1, filename)

# Or if you do not have a variable for your figure
mpld3.save_html(plt.gcf(), filename)
```

PLAY TIME!

To tell you I'm sorry for breaking your heart.