# Astropy

by


## Kaustubh Vaghmare

(IUCAA, Pune)

E-mail: kaustubh[at]iucaa[dot]ernet[dot]in

# A Few Years Ago ...

Several independent developers were developing specific tools for Python...

- PyFITS - enable use of Python to handle FITS files.
- PyWCS - enable World Coordinate System transformations
- atpy / asciitable - handle tables of all kinds
- Some cosmological calculators.
- Some coordinate transformation tools.
- and more...

The problem: Different styles, repeated efforts, no coordination.

The solution: Astropy - unite all efforts under one banner!

# What we will cover?

- Table management features.
- Handling FITS files.
- WCS operations.

# Table Management in Python

If we chose to stay behind by an year or more, the following modules would have been discussed today.

- asciitable
- atpy

But today, we shall cover the "table" sub-module inside Astropy.

# ”atpy” and ”asciitable” are no longer developed.

They have been absorbed by the astropy core package.

But you must still have them installed.

- Some codes you are given may be based on them.
- Some modules may require them.

But while learning, you must learn the astropy versions namely

- astropy.io.ascii
- astropy.table

# astropy.io.ascii vs. astropy.table

- astropy.io.ascii is meant purely for reading and writing tables.
- Is a collection of "extensible" classes which can be extended to support newer formats.

astropy.table

- builds upon io.ascii using its functionality for reading / writing tables
- and adding its own powerful table operations.

You won't need to read much about io.ascii unless your tables have some special outstanding features.

# In Brief - The "Class" Concept

We have discussed the concept of an "object" earlier.

- Objects have well defined behavior.
- They have methods which help you perform supported operations on them.
- Where are all these rules defined?

A "class" is crudely put, a definition which allows one to create objects.

To create table objects, we will need a Table class.

# Let's Start

In [2]:

```
# First we need the Table class to create table objects.
# The warning that will be flashed has so far not affected
# any functional features of Table class
from astropy.table import Table
```

```
/usr/local/lib/python2.7/dist-packages/IPython/zmq/__init__.py:
65: RuntimeWarning: libzmq 4 detected.
        It is unlikely that IPython's zmq code will work proper
ly.
        Please install libzmq stable, which is 2.1.x or 2.2.x
  RuntimeWarning)
```

In [3]:

```
# Next we need to create the Table object using a file.
demo_table = Table.read("demo.txt", format = "ascii")
```

# What if the table does not load?

If you get errors when using read() method, it means that your file is formatted in a way that the standard parser is unable to understand the structure of your file.

What to do? Understand the io.ascii.read() method in detail and supply the various options to Table.read().

eg. `header_start = ";"` or `delimiter="|"` ,etc.

# Displaying Tables.

In [4]:
```
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [5]:

```
demo_table.pprint() # Does exactly the same thing.
# but you can supply options such as
# max_lines, max_width, show_unit, show_name
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [6]:

```
# In this example, we are suppressing column names from appearin
g.
demo_table.pprint(show_name=False)
```

```
 M31 2012-01-02 17.0 17.5
 M31 2012-01-02 17.1 17.4
M101 2012-01-02 15.1 13.5
 M82 2012-02-14 16.2 14.5
 M31 2012-02-14 16.9 17.3
 M82 2012-02-14 15.2 15.5
M101 2012-02-14 15.0 13.6
 M82 2012-03-26 15.7 16.5
M101 2012-03-26 15.1 13.5
M101 2012-03-26 14.8 14.3
```

# More Ways to Print Tables.

Using an interactive table scrolling tool.

```
demo_table.more()
```

Or display it as a formatted table in a browser.

```
demo_table.show_in_browser()
```

# Quickly Check Basic Properties of Loaded Table

In [7]:
```
print len(demo_table) # Number of rows.
```

```
10
```

In [8]:
```
print demo_table.colnames # The names of the columns.
```

```
['name', 'obs_date', 'mag_b', 'mag_v']
```

You can also print any meta information, if available.

```
demo_table.meta
```

# Accessing Columns of the Table

In [9]:
```
print demo_table["name"] # one column
```

```
name
----
 M31
 M31
M101
 M82
 M31
 M82
M101
 M82
M101
M101
```

In [10]:
```
print demo_table["name", "mag_b"] # more than one column
```

```
name mag_b
---- -----
 M31  17.0
 M31  17.1
M101  15.1
 M82  16.2
 M31  16.9
 M82  15.2
M101  15.0
 M82  15.7
M101  15.1
M101  14.8
```

# Accessing Rows in a Table

In [11]:
```
print demo_table[0] # SADLY, row objects do not support printing
.
```

```
<Row 0 of table
 values=('M31', '2012-01-02', 17.0, 17.5)
 dtype=[('name', 'S4'), ('obs_date', 'S10'), ('mag_b', '<f8'),
('mag_v', '<f8')]>
```

In [12]:
```
demo_table[0].data # is one way to get values in a row.
```

Out[12]:
```
('M31', '2012-01-02', 17.0, 17.5)
```

In [13]:
```
lines = demo_table.pformat() # a list of strings, each string a
row, includes header.
print lines[2]
```

```
 M31 2012-01-02  17.0  17.5
```

# Individual Element Access

In [14]:
```
demo_table["name"][0]
```

Out[14]:
```
'M31'
```

In [15]:
```
demo_table[0]["name"] # also works the same as above.
```

Out[15]:
```
'M31'
```

# Sub-sectioning Tables

In [16]:
```
subsection_col = demo_table["name","mag_b"] # by column.
```

In [17]:
```
subsection_row = demo_table[2:5] # by rows.
```

In [18]:
```
subsection_row2 = demo_table[ [1,5,3] ]
```

In [19]:
```
subsection_both = demo_table["name", "mag_b"] [1:5]
```

# Changing elements inside a Table

- You know how to access columns, rows and individual elements.
- Using = sign, you can assign the selected col, row or element another value.

So,

```
demo_table["name"] = ... list of 10 names
demo_table["name"] = "SingleName"
```

will both work.

In [20]:
```
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [21]:
```
demo_table["name"] = "X"
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
   X 2012-01-02  17.0  17.5
   X 2012-01-02  17.1  17.4
   X 2012-01-02  15.1  13.5
   X 2012-02-14  16.2  14.5
   X 2012-02-14  16.9  17.3
   X 2012-02-14  15.2  15.5
   X 2012-02-14  15.0  13.6
   X 2012-03-26  15.7  16.5
   X 2012-03-26  15.1  13.5
   X 2012-03-26  14.8  14.3
```

# Adding New Columns

In [22]:
```
# Method 1
demo_table["NewColumn"] = range(len(demo_table))
print demo_table
```

```
name  obs_date  mag_b mag_v NewColumn
---- ---------- ----- ----- ---------
   X 2012-01-02  17.0  17.5         0
   X 2012-01-02  17.1  17.4         1
   X 2012-01-02  15.1  13.5         2
   X 2012-02-14  16.2  14.5         3
   X 2012-02-14  16.9  17.3         4
   X 2012-02-14  15.2  15.5         5
   X 2012-02-14  15.0  13.6         6
   X 2012-03-26  15.7  16.5         7
   X 2012-03-26  15.1  13.5         8
   X 2012-03-26  14.8  14.3         9
```

In [23]:
```
# Method 2, using Column Object
from astropy.table import Column
newcol = Column( data = range(len(demo_table)), name = "NewColN"
)
demo_table.add_column( newcol, index = 0)
print demo_table
```

```
NewColN name  obs_date  mag_b mag_v NewColumn
------- ---- ---------- ----- ----- ---------
      0    X 2012-01-02  17.0  17.5         0
      1    X 2012-01-02  17.1  17.4         1
      2    X 2012-01-02  15.1  13.5         2
      3    X 2012-02-14  16.2  14.5         3
      4    X 2012-02-14  16.9  17.3         4
      5    X 2012-02-14  15.2  15.5         5
      6    X 2012-02-14  15.0  13.6         6
      7    X 2012-03-26  15.7  16.5         7
      8    X 2012-03-26  15.1  13.5         8
      9    X 2012-03-26  14.8  14.3         9
```

# Removing Columns

In [24]:
```
demo_table.remove_columns(["NewColN", "NewColumn"])
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
   X 2012-01-02  17.0  17.5
   X 2012-01-02  17.1  17.4
   X 2012-01-02  15.1  13.5
   X 2012-02-14  16.2  14.5
   X 2012-02-14  16.9  17.3
   X 2012-02-14  15.2  15.5
   X 2012-02-14  15.0  13.6
   X 2012-03-26  15.7  16.5
   X 2012-03-26  15.1  13.5
   X 2012-03-26  14.8  14.3
```

# For Rows

Similar functions exist. Please read documentation for details. Or explore using iPython.

```
demo_table.remove_row(5)
demo_table.remove_rows( [5,6])
demo_table.remove_rows( slice(3,6) )
```

# Table Sorting

In [25]:

```
demo_table = Table.read("demo.txt", format="ascii")
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
M101 2012-01-02  15.1  13.5
 M82 2012-02-14  16.2  14.5
 M31 2012-02-14  16.9  17.3
 M82 2012-02-14  15.2  15.5
M101 2012-02-14  15.0  13.6
 M82 2012-03-26  15.7  16.5
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

In [26]:
```
demo_table.sort(["name", "mag_b"]) # sort by name, then mag_b
```

In [27]:
```
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
M101 2012-03-26  14.8  14.3
M101 2012-02-14  15.0  13.6
M101 2012-01-02  15.1  13.5
M101 2012-03-26  15.1  13.5
 M31 2012-02-14  16.9  17.3
 M31 2012-01-02  17.0  17.5
 M31 2012-01-02  17.1  17.4
 M82 2012-02-14  15.2  15.5
 M82 2012-03-26  15.7  16.5
 M82 2012-02-14  16.2  14.5
```

In [28]:

```
demo_table.reverse() # Reverse existing table. Descending order!
print demo_table
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
 M82 2012-02-14  16.2  14.5
 M82 2012-03-26  15.7  16.5
 M82 2012-02-14  15.2  15.5
 M31 2012-01-02  17.1  17.4
 M31 2012-01-02  17.0  17.5
 M31 2012-02-14  16.9  17.3
M101 2012-03-26  15.1  13.5
M101 2012-01-02  15.1  13.5
M101 2012-02-14  15.0  13.6
M101 2012-03-26  14.8  14.3
```

# Table Groups

- It is possible to organize the table into groups.
- For example, all entries for object M101 can be selected as a single group.
- One can access individual groups for various operations.
- Also supported "group-wise reductions"

In [29]:
```
demo_table = Table.read("demo.txt", format="ascii")
grouped_table = demo_table.group_by("name")
```

In [30]:
```
# To access groups.
print grouped_table.groups[0] # first group
```

```
name  obs_date  mag_b mag_v
---- ---------- ----- -----
M101 2012-01-02  15.1  13.5
M101 2012-02-14  15.0  13.6
M101 2012-03-26  15.1  13.5
M101 2012-03-26  14.8  14.3
```

# Group-wise Reductions (eg. group-wise mean)

In [31]:

```
import numpy
grouped_table.groups.aggregate( numpy.mean)
```

```
WARNING:astropy:Cannot aggregate column 'obs_date'
```

```
WARNING: Cannot aggregate column 'obs_date' [astropy.table.grou
ps]
```

Out[31]:

| name | mag_b | mag_v |
|---|---|---|
| M101 | 15.0 | 13.725 |
| M31 | 17.0 | 17.4 |
| M82 | 15.7 | 15.5 |

# Filters

- Define a function `some_filter( TableObject, KeyColumns )`.
- The function return True or False.
- Then use the function to remove rows which satisfy some condition.

eg. write a filter to select rows whose mean is positive.

```
def positive_mean( table, key_colnames) :
if np.mean( table["ColName"] > 0:
    return True
else
    return False

t_positive_mean = t_grouped.groups.filter( positive_mean )
```

# Stuff For You To Explore On Your Own

## Stacks - vstack, hstack

## ”joins”

# FITS Files in Python

Again, if this talk was being given few years ago, we would cover

## PyFITS

But today,

## astropy.io.fits

# First step, import the (sub) module.

In [32]:
```
from astropy.io import fits
```

If you have a lot of code that uses PyFits you can say,

```
import astropy.io.fits as pyfits
```

or whatever alias you use and a lot of PyFITS based code should work fine.

Next step, open a FITS file. The method used for this creates a hdulist object. HDU = Header Data Unit

In [33]:
```
hdulist = fits.open("example.fits")
```

Next, check up some basic information about the FITS file.

In [34]:
```
hdulist.info()
```

```
Filename: example.fits
No.    Name         Type      Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU     104   (318, 318)   int16
```

As you can see, this is a single extension FITS file.

# Accessing the header

In [35]:
```
hdulist[0].header
```

Out[35]:
```
SIMPLE  =                    T /FITS header

BITPIX  =                   16 /No.Bits per pixel

NAXIS   =                    2 /No.dimensions

NAXIS1  =                  318 /Length X axis

NAXIS2  =                  318 /Length Y axis

DATE    = '06/05/97          ' /Date of FITS file creation

ORIGIN  = 'CASB -- STScI     ' /Origin of FITS image

PLTLABEL= 'E 1398            ' /Observatory plate label

PLATEID = '06C0              ' /GSSS Plate ID

REGION  = 'XE320             ' /GSSS Region Name

DATE-OBS= '18/04/55          ' /UT date of Observation

UT      = '05:55:00.00       ' /UT time of observation

EPOCH   =  1.9552938232422E+03 /Epoch of plate

PLTRAH  =                   12 /Plate center RA
```




```
PLTRAM  =                   15 /
```

# Specific stuff within header.

In [36]:
```
hdulist[0].header["NAXIS1"] # by header keyword
```

Out[36]:
```
318
```

In [37]:
```
hdulist[0].header[1] # or by header number.
```

Out[37]:
```
16
```

In [38]:
```
all_keys = hdulist[0].header.keys() # get a list of all keys.
```

In [39]:
```
all_values = hdulist[0].header.values()
```

You can also change the header values as if it were a dictionary.

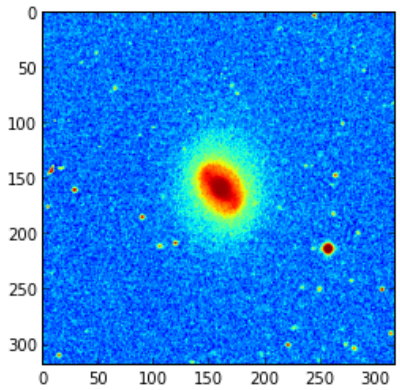
# Now, the data

In [40]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [41]:
```
imshow( np.log10(hdulist[0].data) )
```

Out[41]:
```
<matplotlib.image.AxesImage at 0x49fa250>
```



# Axis Conventions

If you load a FITS image in Python, in FORTRAN/C or in ds9, the image viewer, what does I(X,Y) can give you different results!!!

There is a different in whether the following code moves along horizontal axis first or vertical axis first.

```
for x in range(header["NAXIS1"]):
    for y in range(header["NAXIS2"]):
        ...
```

If you ask me - my answer - **I don't know! I am always confused.**

My strategy: **Use iPython to load the image. Also load image in ds9. Do a bit of fiddling around and write your loops!**

# Writing FITS files

If you have a HDUlist object, you simply say,

```
hdulist.writeto("name.fits")
```

If you want to make a file from scratch, create a dictionary of headers and the data array.

```
primaryhdu = fits.PrimaryHDU(data, header)
primaryhdu.writeto("something.fits")
```

# World Coordinate Systems

Few years ago,

```
import pywcs
```

In the era of Astropy,

```
from astropy import wcs
```

Funtionally, they are more or less the same.

# Create a WCS object.

In [42]:
```python
from astropy import wcs
w = wcs.WCS("wcsdemo.fits")
```

```
WARNING:astropy:The following header keyword is invalid or foll
ows an unrecognized non-standard convention:
CO1_3   =-3.30161034511646E-06

WARNING:astropy:The following header keyword is invalid or foll
ows an unrecognized non-standard convention:
CO1_4   =-2.55990918514719E-11

WARNING:astropy:The following header keyword is invalid or foll
ows an unrecognized non-standard convention:
CO1_5   =-7.84276839450913E-12

WARNING:astropy:The following header keyword is invalid or foll
ows an unrecognized non-standard convention:
CO2_1   =-9.80116357595331E-05

WARNING:astropy:The following header keyword is invalid or foll
ows an unrecognized non-standard convention:
CO2_2   =-3.32208424969949E-06

WARNING:astropy:FITSFixedWarning: RADECSYS= 'FK5 '
RADECSYS is non-standard, use RADESYSa.

WARNING: The following header keyword is invalid or follows an
unrecognized non-standard convention:
CO1_3   =-3.30161034511646E-06
                    [astropy.io.fits.card]
WARNING: The following header keyword is invalid or follows an
```

While the above is allowed, taking into account that FITS files can have multiple extensions, you should,

In [43]:
```
hdulist = fits.open("wcsdemo.fits")
w = wcs.WCS(hdulist[0].header)
```

It's thw WCS object which has methods to perform any coordinate transformations.

In [44]:
```
w.wcs_pix2sky(1000, 2000, 1)
```

Out[44]:
```
[array(184.95026582155023), array(1.383830415506317)]
```

In [45]:
```
w.all_pix2sky(1000, 2000, 1)
```

Out[45]:
```
[array(184.95026582155023), array(1.383830415506317)]
```

- Which pixel? (1000, 2000) or (1001, 2001). It's (1000,2000), the third argument 1 assures you that.
- Difference between wcs_pix2sky and all_pix2sky - the latter takes into account some higher order transformations / corrections into account.
- Output? (RA, DEC) in degrees.

To do a reverse transformation.

In [46]:
```
w.wcs_sky2pix(184.6, 1.38, 1)
```

Out[46]:
```
[array(1503.8827051960322), array(1996.7722524018488)]
```

In [47]:
```
w.calcFootprint() # The four corners of an image.
```

Out[47]:
```
array([[  1.85650302e+02,  -1.95542013e-03],
       [  1.85638488e+02,   2.84300473e+00],
       [  1.82780624e+02,   2.82938730e+00],
       [  1.82795939e+02,  -1.55561710e-02]])
```

# Cosmology

This submodule of astropy allows you to do various cosmological calculations based on a model of cosmology.

We begin by importing the cosmology sub-module.

In [48]:
```
from astropy import cosmology
```

Now, before we can make do any cosmological calculations, we need to choose a model. Let's do that.

In [49]:
```
print cosmology.parameters.available
```

```
('WMAP5', 'WMAP7', 'WMAP9', 'Planck13')
```

The above are the various models of cosmology available, you can choose one of them by saying,

In [51]:
```
from astropy.cosmology import WMAP9
```

Or you could define your own cosmology by saying,

```
from astropy.comsology import FlatLambdaCDM
mycosmo = FlatLambdaCDM(..., ...., ....)
```

Refer documentation for more details.

# Performing Cosmological Calculations

In [52]:
```
WMAP9.H(1.5) # what is the Hubble parameter at redshift 1.5?
```

Out[52]: $157.973 \ \frac{\mathrm{km}}{\mathrm{Mpc\,s}}$

In [54]:
```
WMAP9.Ode(3)  # density parameter for dark energy at redshift z=
3   (in units of critical density)
```

Out[54]:
```
0.037406961669436807
```

In [55]:
```
WMAP9.critical_density(3)   # critical density at z=3
```

Out[55]: $1.72151 \times 10^{-28} \ \frac{\mathrm{g}}{\mathrm{cm^{3}}}$

In [56]:
```
WMAP9.Tcmb(1000)    # CMB temperature at z=1000
```

Out[56]: $2727.72\ \mathrm{K}$

In [57]:
```
WMAP9.angular_diameter_distance(2)  # Angular diameter distance
in Mpc at z=2.
```

Out[57]: $1763.91\ \mathrm{Mpc}$

In [58]:
```
WMAP9.arcsec_per_kpc_comoving(3)  # Angular separation in arcsec
 corresponding to a comoving kpc at z=3
```

Out[58]: $0.031714\ \frac{''}{\mathrm{kpc}}$

In [59]:
```
WMAP9.scale_factor(4)     # a = 1/(1+z)
```

Out[59]: 0.2

In [60]:
```
WMAP9.age(1000)       # Age of universe at z=1000
```

Out[60]: $0.000434354\ \mathrm{Gyr}$

In [61]:
```
print dir(WMAP9)
```

```
['H', 'H0', 'Neff', 'Ode', 'Ode0', 'Ogamma', 'Ogamma0', 'Ok', '
Ok0', 'Om', 'Om0', 'Onu', 'Onu0', 'Tcmb', 'Tcmb0', 'Tnu', 'Tnu0
', '_H0', '_Neff', '_Ode0', '_Ogamma0', '_Ok0', '_Om0', '_Onu0'
, '_Tcmb0', '_Tnu0', '__abstractmethods__', '__class__', '__del
attr__', '__dict__', '__doc__', '__format__', '__getattribute__
', '__hash__', '__init__', '__metaclass__', '__module__', '__ne
w__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_
abc_cache', '_abc_negative_cache', '_abc_negative_cache_version
', '_abc_registry', '_critical_density0', '_h', '_hubble_distan
ce', '_hubble_time', '_massivenu', '_namelead', '_neff_per_nu',
 '_nmassivenu', '_nmasslessnu', '_nneutrinos', '_tfunc', '_w_in
tegrand', '_xfunc', 'absorption_distance', 'age', 'angular_diam
eter_distance', 'angular_diameter_distance_z1z2', 'arcsec_per_k
pc_comoving', 'arcsec_per_kpc_proper', 'comoving_distance', 'co
moving_transverse_distance', 'comoving_volume', 'critical_densi
ty', 'critical_density0', 'de_density_scale', 'distmod', 'efunc
', 'h', 'has_massive_nu', 'hubble_distance', 'hubble_time', 'in
v_efunc', 'kpc_comoving_per_arcmin', 'kpc_proper_per_arcmin', '
lookback_time', 'luminosity_distance', 'm_nu', 'name', 'nu_rela
tive_density', 'scale_factor', 'w']
```