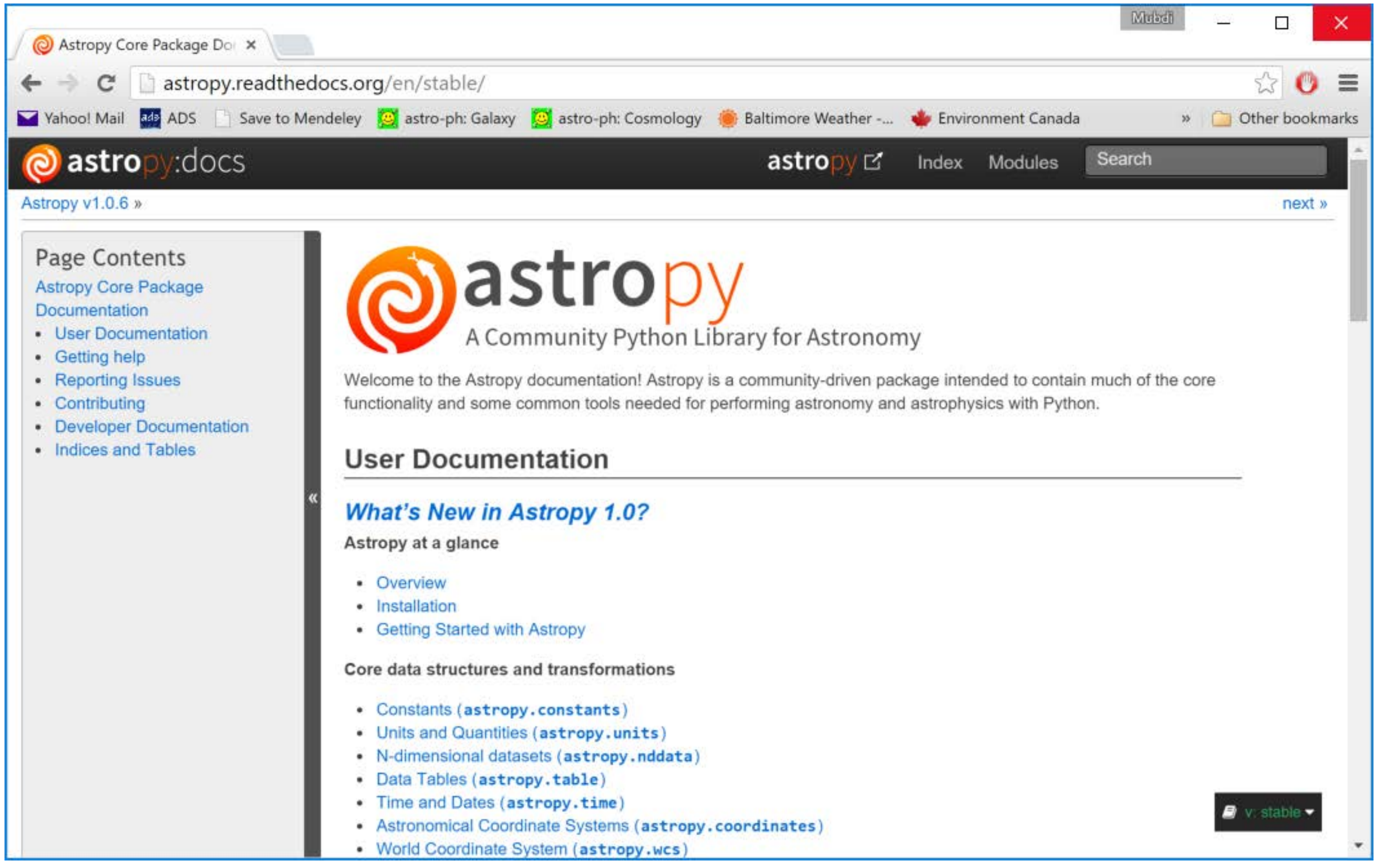
3. NUMERICAL METHODS II

JHU Physics & Astronomy Python Workshop 2015

Lecturer: Mubdi Rahman

INTRODUCING ASTROPY!



Page Contents

- Astropy Core Package Documentation
  - User Documentation
  - Getting help
  - Reporting Issues
  - Contributing
  - Developer Documentation
  - Indices and Tables

astropy

A Community Python Library for Astronomy

Welcome to the Astropy documentation! Astropy is a community-driven package intended to contain much of the core functionality and some common tools needed for performing astronomy and astrophysics with Python.

User Documentation

What's New in Astropy 1.0?

Astropy at a glance

- Overview
- Installation
- Getting Started with Astropy

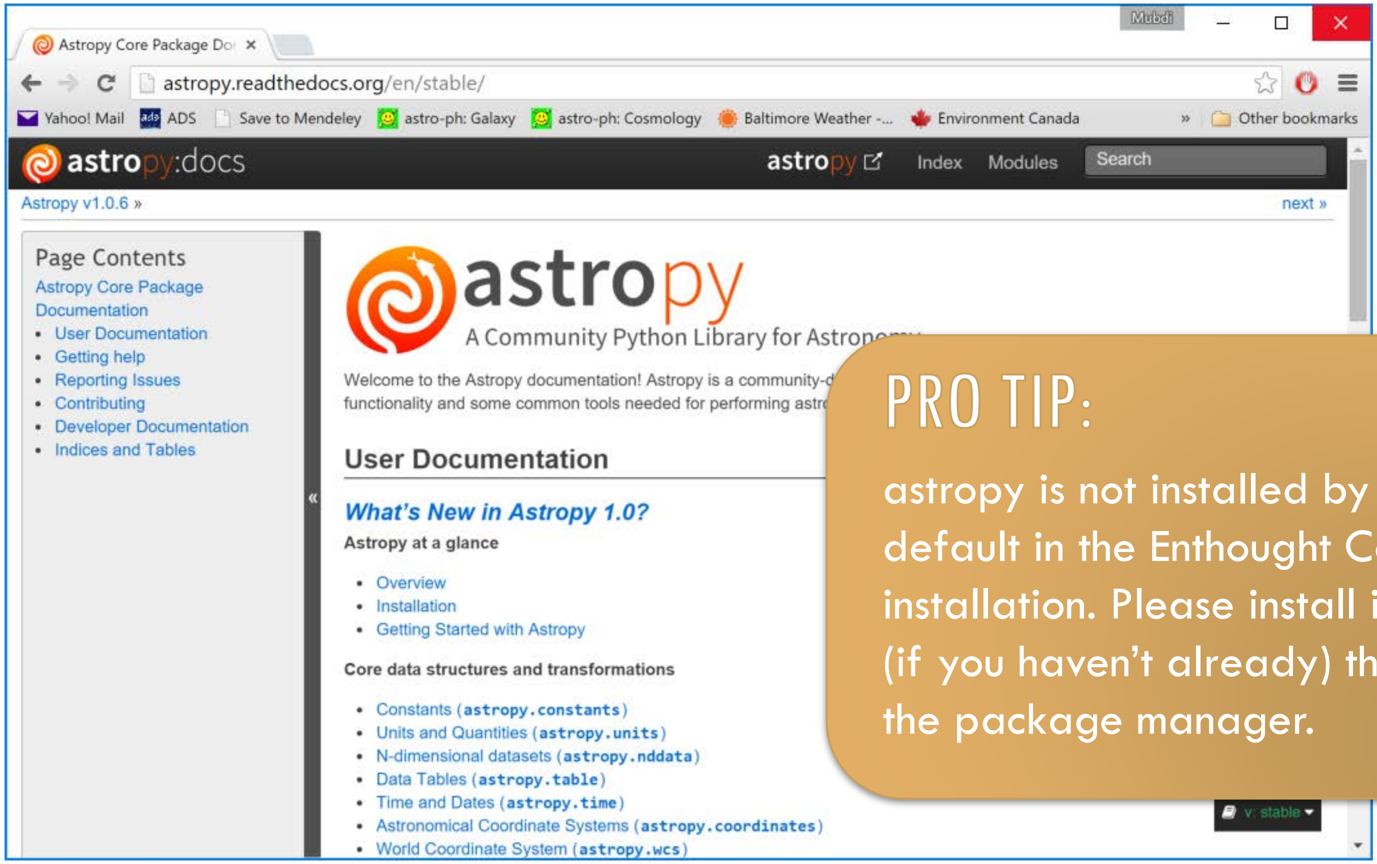
Core data structures and transformations

- Constants (`astropy.constants`)
- Units and Quantities (`astropy.units`)
- N-dimensional datasets (`astropy.nddata`)
- Data Tables (`astropy.table`)
- Time and Dates (`astropy.time`)
- Astronomical Coordinate Systems (`astropy.coordinates`)
- World Coordinate System (`astropy.wcs`)

Contains lots 'o useful astronomical functionality.

The Docs: http://astropy.readthedocs.org/en/stable/

INTRODUCING ASTROPY!


Astropy Core Package Do ×

astropy.readthedocs.org/en/stable/

Yahoo! Mail · ADS · Save to Mendeley · astro-ph: Galaxy · astro-ph: Cosmology · Baltimore Weather -... · Environment Canada · » · Other bookmarks

astropy:docs · astropy · Index · Modules · Search

Astropy v1.0.6 » · next »

Page Contents

Astropy Core Package Documentation

- User Documentation
- Getting help
- Reporting Issues
- Contributing
- Developer Documentation
- Indices and Tables

astropy

A Community Python Library for Astrono

Welcome to the Astropy documentation! Astropy is a community-d
functionality and some common tools needed for performing astr

User Documentation

What's New in Astropy 1.0?

Astropy at a glance

- Overview
- Installation
- Getting Started with Astropy

Core data structures and transformations

- Constants (`astropy.constants`)
- Units and Quantities (`astropy.units`)
- N-dimensional datasets (`astropy.nddata`)
- Data Tables (`astropy.table`)
- Time and Dates (`astropy.time`)
- Astronomical Coordinate Systems (`astropy.coordinates`)
- World Coordinate System (`astropy.wcs`)

v: stable

PRO TIP:

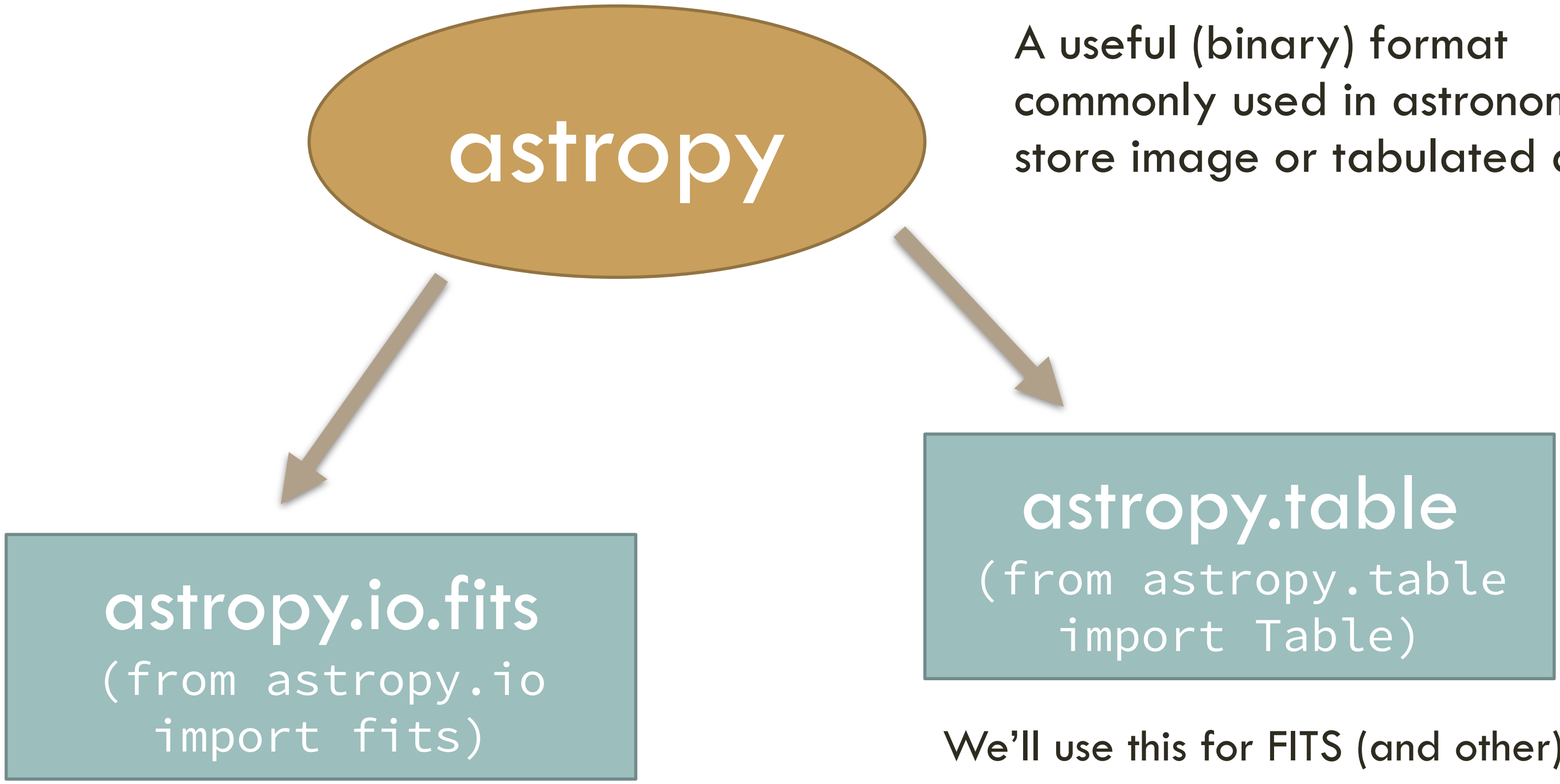
astropy is not installed by default in the Enthought Canopy installation. Please install it now (if you haven't already) through the package manager.

Contains lots 'o useful astronomical functionality.

The Docs: http://astropy.readthedocs.org/en/stable/

FITS FILES!

A useful (binary) format commonly used in astronomy to store image or tabulated data.



```
(from astropy.io
import fits)
```

```
(from astropy.table
import Table)
```

We'll use this for FITS images

We'll use this for FITS (and other) tables

FITS FILES!

astropy

A useful (binary) format commonly used in astronomy to store image or tabulated data.

astropy.table

`(from astropy.table import Table)`

We'll use this for FITS (and other) tables

PRO TIP:

When you import something with a capital letter first (i.e., from astropy.table import Table), you're importing a class. These are special types of variables with useful *methods*

FITS FILES!

astropy

A useful (binary) format commonly used in astronomy to store image or tabulated data.

astropy.io.fits

```
(from astropy.io
import fits)
```

We'll use this for FITS images

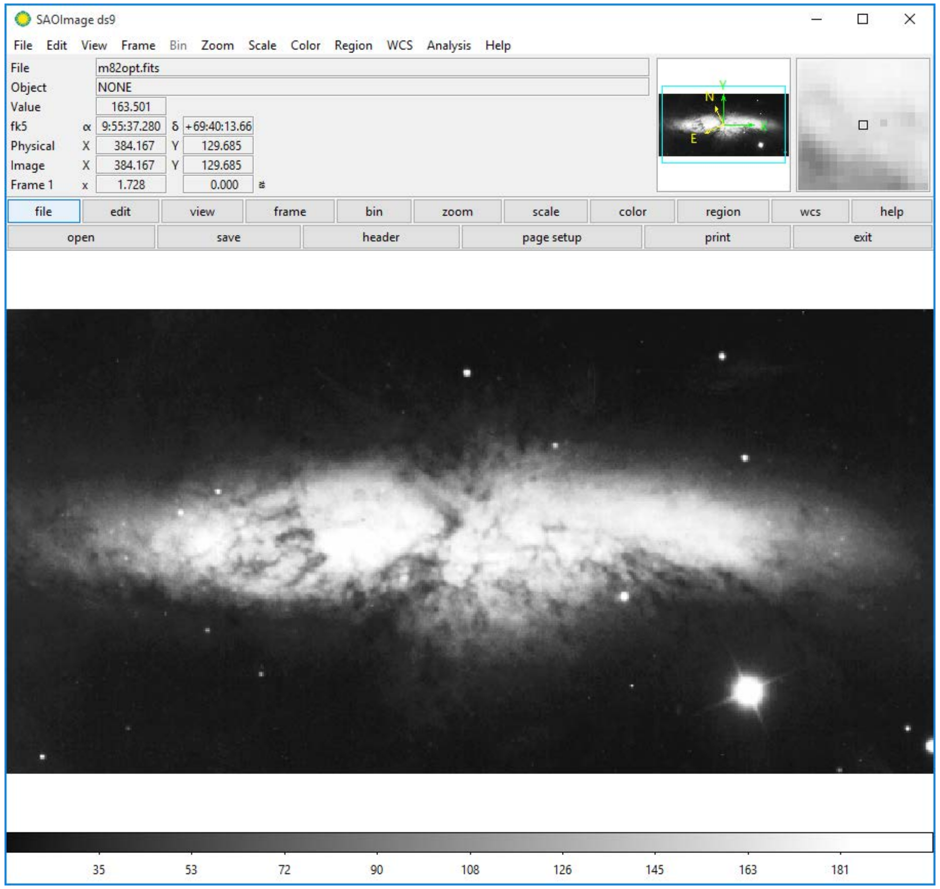
PRO TIP 2:

You can also deal with tables through the normal astropy.io.fits interface. The "table" interface is quite slick, however and makes life easier (especially when making new tables).

e

ther)

FITS IMAGES


SAOImage ds9
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
file edit view frame bin zoom scale color region wcs help
open save header page setup print exit
N
E
X
Y
35 53 72 90 108 126 145 163 181

File m82opt.fits
Object NONE
Value 163.501
fk5 α 9:55:37.280 δ +69:40:13.66
Physical X 384.167 Y 129.685
Image X 384.167 Y 129.685
Frame 1 x 1.728 0.000

A FITS file open in DS9 (a common viewer)

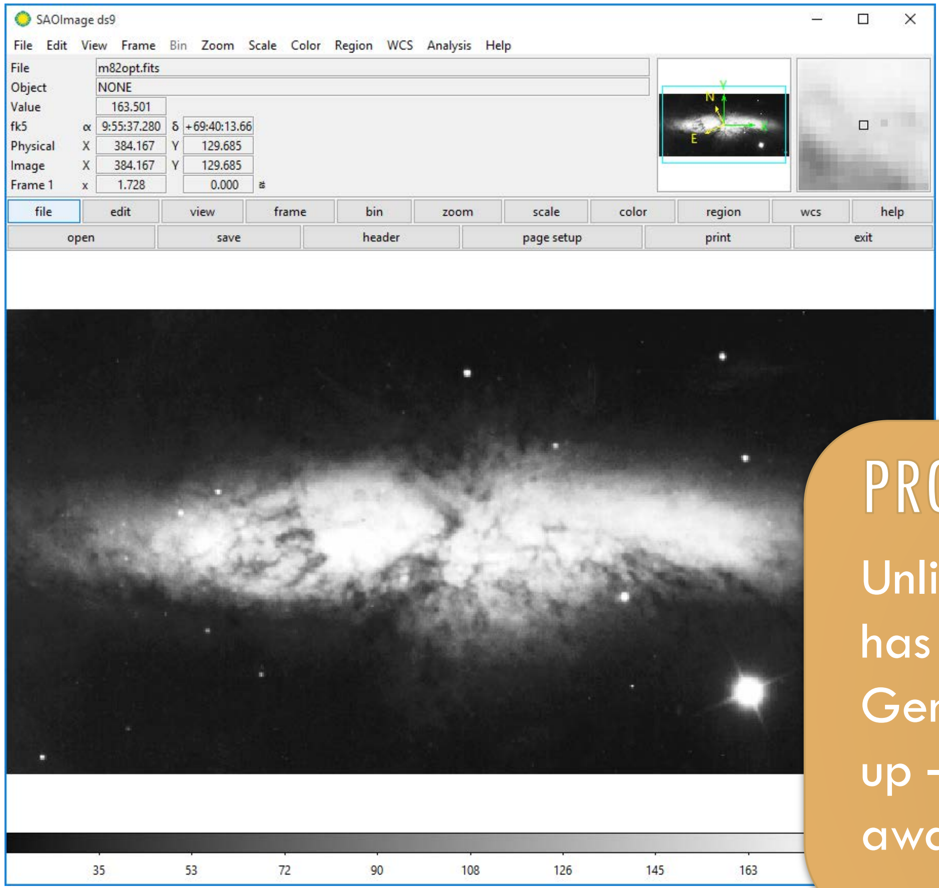
FITS files can store multidimensional data (commonly 2 or 3 dimensions).

Any given FITS file can contain multiple images (or tables) called **extensions**

Every FITS extension contains a **header** and **data**.

FITS headers can contain World Coordinate System (wcs) information that indicates where a given pixel is on the sky

FITS IMAGES


SAOImage ds9
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
file edit view frame bin zoom scale color region wcs help
open save header page setup print exit
N
E
35 53 72 90 108 126 145 163

File: m82opt.fits
Object: NONE
Value: 163.501
fk5 α 9:55:37.280 δ +69:40:13.66
Physical X 384.167 Y 129.685
Image X 384.167 Y 129.685
Frame 1 x 1.728 0.000

A FITS file open in DS9 (a common viewer)

FITS files can store multidimensional data (commonly 2 or 3 dimensions).

Any given FITS file can contain multiple images (or tables) called **extensions**

ntains

in
em

pixel

PRO TIP:

Unlike python, FITS convention has indexing starting at 1. Generally astropy covers this up – but you should be aware of this.

READING IN FITS IMAGES

Convenience functions make reading FITS images easy:

```
from astropy.io import fits
img1 = fits.getdata(filename) # Getting the image
head1 = fits.getheader(filename) # and the Header
```

This opens the image as a Numpy array, and the header as a "dictionary-like" object (i.e., you can access the individual header keywords through "`head1['key']`").

To open other extensions in the fits file:

```
img1 = fits.getdata(filename, 0) # Primary Ext
img2 = fits.getdata(filename, 1) # Second Ext
img2 = fits.getdata(filename, ext=1) # Equivalent
```

READING IN FITS IMAGES

Convenience functions make reading FITS images easy:

```
from astropy.io import fits
img1 = fits.getdata(filename) # Getting the image
head1 = fits.getheader(filename) # and the Header
```

This opens the image as a Numpy array, and the header as a "dictionary-like" object (i.e., you can access the individual header keywords through "`head1['key']`").

To open other extensions in the fits file:

```
img1 = fits.getdata(filename,
img2 = fits.getdata(filename,
img2 = fits.getdata(filename,
```

PRO TIP:

In addition to local files, you can open FITS files on the internet by using the url as opposed to the file name.

READING IN FITS IMAGES

Convenience functions make reading FITS images easy:

```
from astropy.io import fits
img1 = fits.getdata(filename) # Getting the image
head1 = fits.getheader(filename) # and the Header
```

This opens the image as a Numpy array,
“dictionary-like” object (i.e., you can acc
keywords through “`head1['key']`”)

To open other extensions in the fits file:

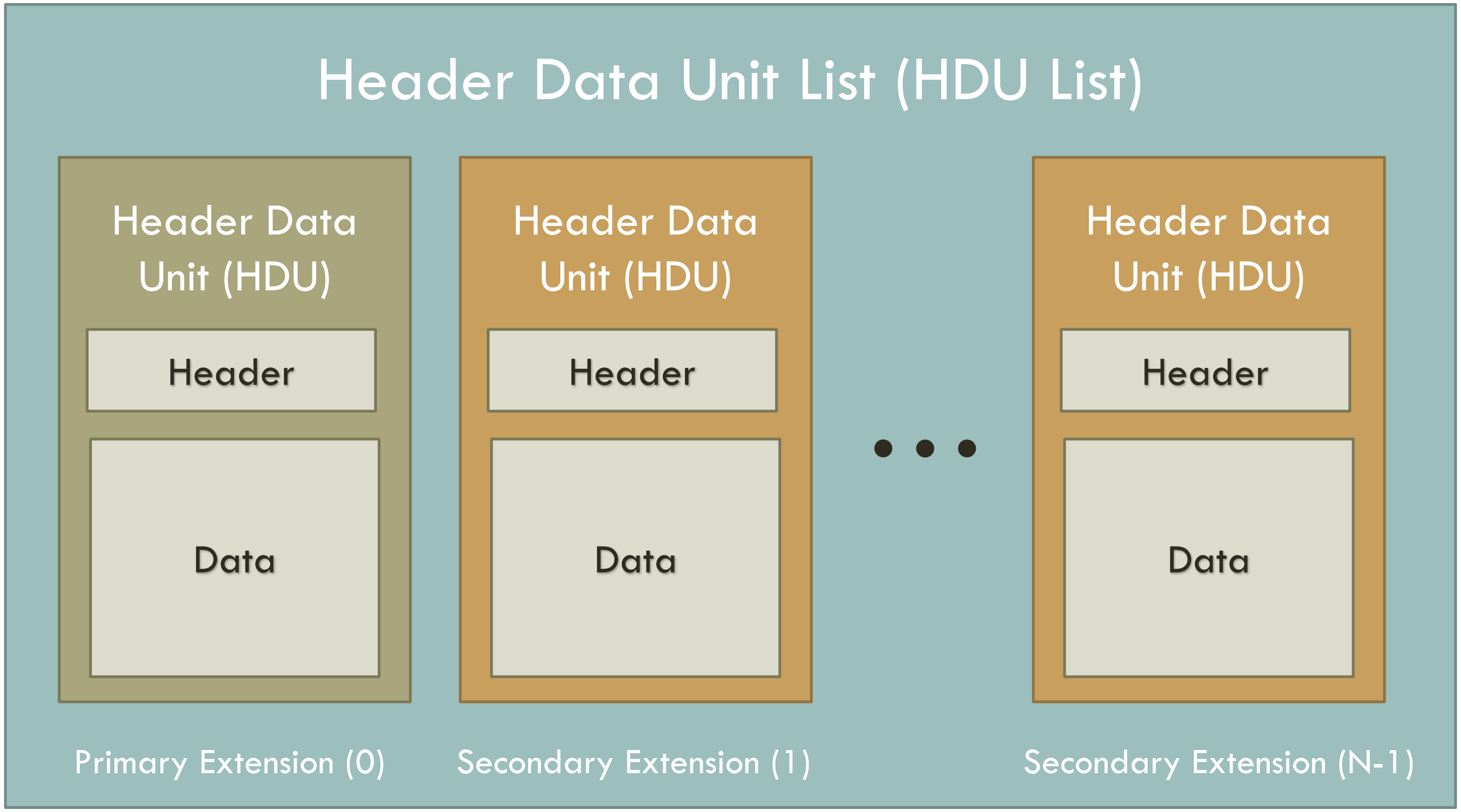
```
img1 = fits.getdata(filename,
img2 = fits.getdata(filename,
img2 = fits.getdata(filename,
```

PRO TIP 2:

This is **not** the most efficient way to open a FITS file, especially larger ones. If you want to manipulate large data sets multiple times, there’s a faster way.

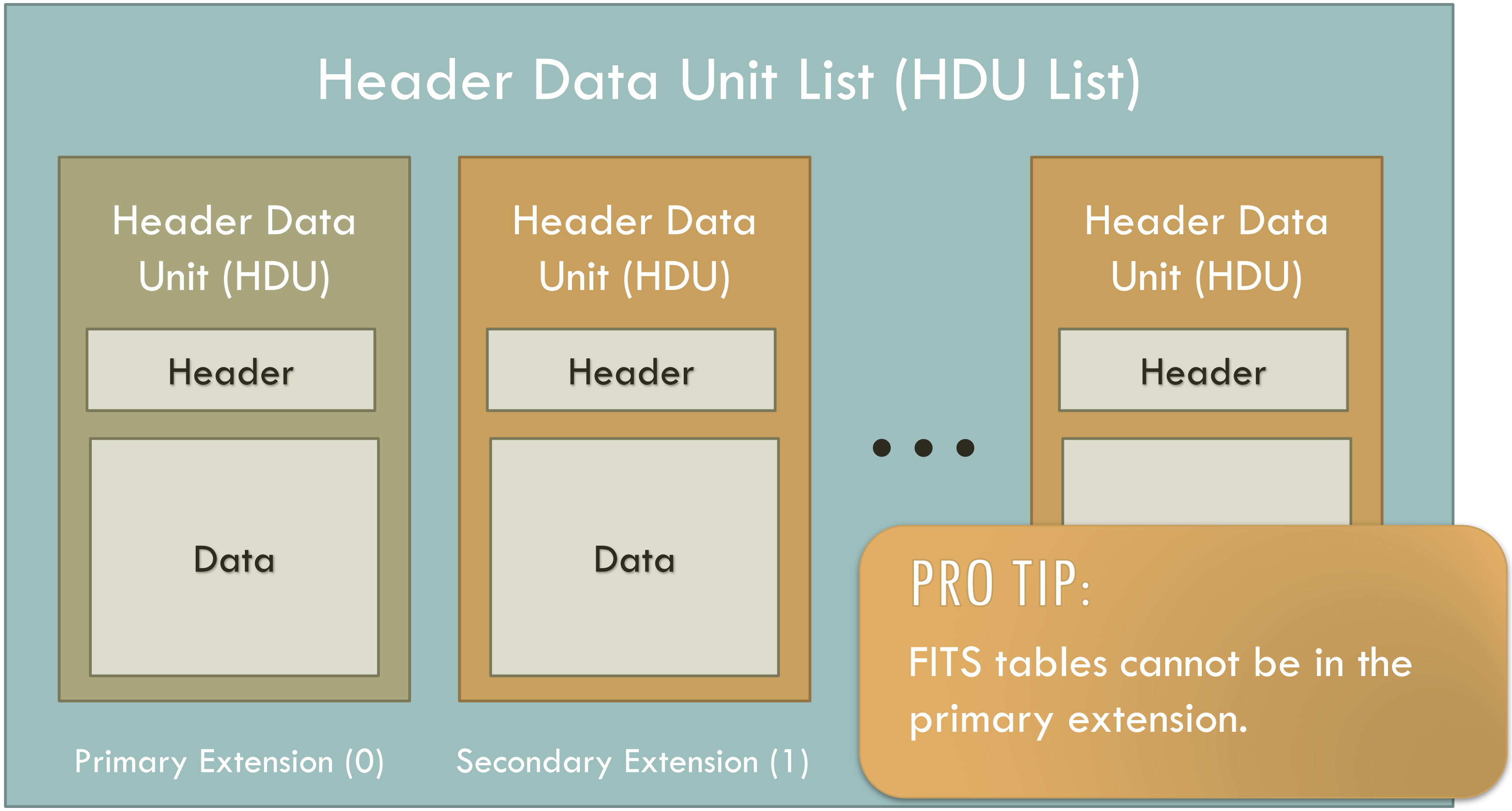
FITS FILES: A MORE TECHNICAL REVIEW

Basic structure of a FITS file:



FITS FILES: A MORE TECHNICAL REVIEW

Basic structure of a FITS file:



PRO TIP:

FITS tables cannot be in the primary extension.

READING IN A FITS FILE (EXPANDED)

Reading a file, now knowing what a FITS file consists of:

```
hdulist = fits.open(filename) # Getting the HDUlist
hdulist.info() # The composition of the file
```

Now getting the header and/or data:

```
head0 = hdulist[0].header # Primary Ext Header
data0 = hdulist[1].data # Second Ext Data
```

Writing to a new file and closing:

```
hdulist.writeto(filename)
hdulist.close() # Closing the FITS file
```

READING IN A FITS FILE (EXPANDED)

Reading a file, now knowing what a FITS file consists of:

```
hdulist = fits.open(filename) # Getting the HDUlist
hdulist.info() # The composition of the file
```

Now getting the header and/or data:

```
head0 = hdulist[0].header # Prim
data0 = hdulist[1].data # Second
```

Writing to a new file and closing:

```
hdulist.writeto(filename)
hdulist.close() # Closing the FIT
```

PRO TIP:

FITS files are read in such that the first axis (often the RA for astronomical images) is read in as the last axis in the numpy array. Be sure to double check that you have the axis you need.

READING IN A FITS FILE (EXPANDED)

Reading a file, now knowing what a FITS file consists of:

```
hdulist = fits.open(filename) # Getting the HDUlist
hdulist.info() # The composition of the file
```

Now getting the header and/or data:

```
head0 = hdulist[0].header # Prima
data0 = hdulist[1].data # Second
```

Writing to a new file and closing:

```
hdulist.writeto(filename)
hdulist.close() # Closing the FIT
```

PRO TIP 2:

writeto will, by default, fail if you try to overwrite an existing file. To force an overwrite, pass the clobber argument:

```
clobber = True
```

WRITING OUT A FITS IMAGE

Making a new FITS image is also easy from a Numpy array:

```
# Making a Primary HDU (required):
primaryhdu = fits.PrimaryHDU(arr1) # Makes a header
# or if you have a header that you’ve created:
primaryhdu = fits.PrimaryHDU(arr1, header=head1)

# If you have additional extensions:
secondhdu = fits.ImageHDU(arr2)

# Making a new HDU List:
hdulist1 = fits.HDUList([primaryhdu, secondhdu])

# Writing the file:
hdulist1.writeto(filename, clobber=True)
```

SHORT DETOUR: GLOB MODULE

In one of the many useful python packages, **glob** lets you get lists of files using wildcards:

```
import glob

# Getting list of all files in current directory:
filelist1 = glob.glob('*') # or
filelist1 = glob.glob('./*')

# Getting list of all files in all directories two
levels down with the extension '.fits':
filelist2 = glob.glob('*/*/*.fits')
```

SHORT DETOUR: OS MODULE

Additionally, the **os** module provides a large number of useful filesystem functions:

```
import os

# Basic File Operations:
os.remove(filename) # Delete file named filename
os.rename(oldfilename, newfilename) # Rename file
os.mkdir(dirname) # Making new directory

# Path functions:
os.path.exists(loc) # Checks if loc exists
# Splits loc into directory and file
os.path.split(loc)
# Splits loc into path+file and extension
os.path.splitext(loc)
```

SHORT DETOUR: LAMBDA FUNCTIONS

Sometimes you want to define a simple function without the full function syntax. **Lambda functions** exist for this exact reason:

```
# Defining the Function:
funct1 = lambda x: x**2 # Returns the square of x

# Using the Function:
tmpvar1 = funct1(5)

# Can use multiple variables:
funct2 = lambda x,y: x + y

# Using the Function:
tmpvar2 = funct2(5, 6)
```

TABLES (& FITS TABLES)

While you can use the FITS interface to open tables, Astropy makes it very easy and convienient with the **astropy.table** interface:

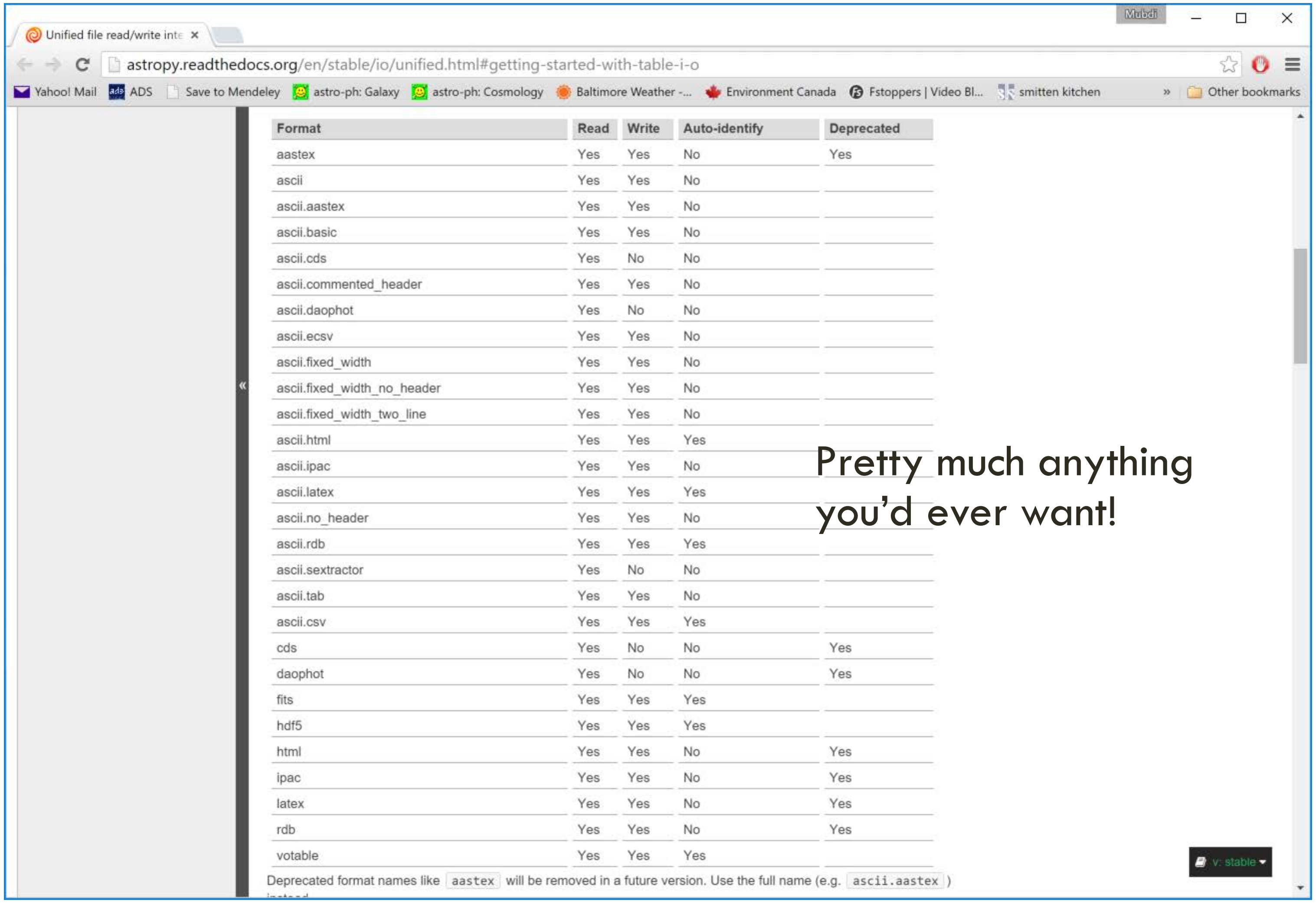
```
from astropy.table import Table

# Getting the first table
t1 = Table.read(filename.fits)

# Getting the second table
t2 = Table.read(filename.fits, hdu=2)
```

This provides a *really* flexible **Table** object that is a pleasure to deal with. It is easy to access different types of data, and read in and output to a wide variety of formats (not just FITS)

TABLE FORMATS



Format	Read	Write	Auto-identify	Deprecated
aastex	Yes	Yes	No	Yes
ascii	Yes	Yes	No	
ascii.aastex	Yes	Yes	No	
ascii.basic	Yes	Yes	No	
ascii.cds	Yes	No	No	
ascii.commented_header	Yes	Yes	No	
ascii.daophot	Yes	No	No	
ascii.ecsv	Yes	Yes	No	
ascii.fixed_width	Yes	Yes	No	
ascii.fixed_width_no_header	Yes	Yes	No	
ascii.fixed_width_two_line	Yes	Yes	No	
ascii.html	Yes	Yes	Yes	
ascii.ipac	Yes	Yes	No	
ascii.latex	Yes	Yes	Yes	
ascii.no_header	Yes	Yes	No	
ascii.rdb	Yes	Yes	Yes	
ascii.sextractor	Yes	No	No	
ascii.tab	Yes	Yes	No	
ascii.csv	Yes	Yes	Yes	
cds	Yes	No	No	Yes
daophot	Yes	No	No	Yes
fits	Yes	Yes	Yes	
hdf5	Yes	Yes	Yes	
html	Yes	Yes	No	Yes
ipac	Yes	Yes	No	Yes
latex	Yes	Yes	No	Yes
rdb	Yes	Yes	No	Yes
votable	Yes	Yes	Yes	

Deprecated format names like `aastex` will be removed in a future version. Use the full name (e.g. `ascii.aastex`)

Pretty much anything you'd ever want!

PLAYING WITH TABLE DATA

A table is both a **dictionary-like** and **numpy array-like** data type that can either be accessed by key (for columns) or index (for rows):

```
# Getting column names, number of rows:
t1.colnames, len(t1)

# Getting specific columns:
t1[‘name1’], t1[[‘name1’, ‘name2’]]

# Getting specific rows (all normal indexing works):
t1[0], t1[:3], t1[::-1]

# Where searching also works:
inds = np.where(t1[‘name1’] > 5)
subtable = t1[inds] # Gets all columns
```

PLAYING WITH TABLE DATA

A table is both a **dictionary-like** and **numpy array-like** data type that can either be accessed by key (for columns) or index (for rows):

```
# Getting column names, number of rows:
t1.colnames, len(t1)

# Getting specific columns:
t1['name1'], t1[['name1', 'name2']]

# Getting specific rows (all norm
t1[0], t1[:3], t1[::-1]

# Where searching also works:
inds = np.where(t1['name1'] > 5)
subtable = t1[inds] # Gets all co
```

PRO TIP:

Extracting a single column will give you a Numpy array-like variable with all your favourite methods attached.

MAKING A TABLE

To make a table manually is easy with Numpy arrays:

```
# Given two columns (1D) arr1 and arr2:
t1 = Table([arr1, arr2], names=(“a”, “b”))

# The columns are named “a” and “b”.

# Adding an additional column:
col1 = Table.Column(name=“c”, data=arr3)
t1.add_column(col1)

# Adding an additional row:
row = np.array([1, 2, 3])
t1.add_row(row)
```

WRITING OUT A TABLE

Writing out a table is also quite simple:

```
# Writing out FITS table:
t1.write(filename.fits)

# Writing out specific text type:
t1.write(filename.txt, format=‘ascii.tab’)

# Can even write out to LaTeX:
t1.write(filename.tex, format=‘ascii.latex’)
```

WRITING OUT A TABLE

Writing out a table is also quite simple:

```
# Writing out FITS table:
t1.write(filename.fits)

# Writing out specific text type:
t1.write(filename.txt, format=‘ascii.tab’)

# Can even write out to LaTeX
t1.write(filename.tex, format
```

PRO TIP:

To quickly investigate a table in a nicely formatted manner, you can do:

```
t1.show_in_browser()
```

EXERCISE TIME!

Yes, I'd like to visit the moon, but I don't think I'd like to live there.