An introduction to scientific programming with

python™

**Session 7**:

Python for specialists

An introduction to scientific programming with

**Session 7.1**:

Python for astronomers

# Python for astronomers

- Python is *great* for astronomy
- Support from STScI and many other observatories
- Many instrument reduction packages written in Python
- Lots of other resources…

# Python for astronomers

astropy

A Community Python Library for Astronomy

- recent and ongoing effort to create a uniform package
    - feature-rich, and rapidly becoming more so
    - strong community and observatory support
    - worth supporting and contributing
        - affiliated packages

# astropy

astropy
A Community Python Library for Astronomy

- Astronomical constants, units, times and dates
- Astronomical coordinate systems
- Cosmology calculations
- Virtual Observatory integration
- Astronomy specific additions to numpy/scipy tools:
    - n-dimensional datasets, tables
    - model fitting, convolution, filtering, statistics
- Undergoing rapid development – but mostly stable (v3.x)
- Open source, on GitHub

# Units

- Highly integrated usage of **units** and **quantities**

```
>>> from astropy import units as u

>>> 42.0 * u.meter
<Quantity 42. m>

>>> [1., 2., 3.] * u.m
<Quantity [1., 2., 3.] m>

>>> import numpy as np
>>> np.array([1., 2., 3.]) * u.m
<Quantity [1., 2., 3.] m>
```

# astropy

- Highly integrated usage of **units** and **quantities**

```
>>> distance = 15.1 * u.meter
>>> time = 32.0 * u.second
>>> distance / time
<Quantity 0.471875 m / s>

>>> timescale = (3.0 * u.kilometer /
                 (130.51 * u.meter / u.second))
>>> timescale
<Quantity 0.022986744310780783 km s / m>
>>> timescale.decompose()
<Quantity 22.986744310780782 s>
```

# astropy

- Highly integrated usage of **units** and **quantities**

```
>>> x = 1.0 * u.parsec
>>> x.to(u.km)
<Quantity 30856775814671.914 km>


>>> mag = 17 * u.STmag
>>> mag.to(u.erg/u.s/u.cm**2/u.AA)
<Quantity 5.754399373371e-16 erg / (Angstrom cm2 s)>
>>> mag.to(u.erg/u.s/u.cm**2/u.Hz,
           u.spectral_density(5500 * u.AA))
<Quantity 5.806369586672163e-27 erg / (cm2 Hz s)>
```

# Matching catalogues

- Don't use simple nested loops
  - inefficient, don't handle edge cases, …
- Better to use:
  - **astropy libraries**
  - searchsorted
  - scipy set library methods
  - do it outside of Python (e.g., using TOPCAT or STILTS)

# astropy

astropy
A Community Python Library for Astronomy

- Catalogue matching
    - understands astronomical coordinates
    - fast (uses, and stores, KD tree)
    - **one-to-one**, one-to-many, separations, etc.

```
from astropy.coordinates import SkyCoord
catalogcoord = SkyCoord(ra=ra_list, dec=dec_list)
matchcoord = SkyCoord(ra=ra, dec=dec, frame='FK4')

from astropy.coordinates import match_coordinates_sky
idx, d2d, d3d = match_coordinates_sky(matchcoord, catalogcoord)
# or
idx, d2d, d3d = matchcoord.match_to_catalog_sky(catalogcoord)
```

# astropy

astropy
A Community Python Library for Astronomy

- Catalogue matching
  - understands astronomical coordinates
  - fast (uses, and stores, KD tree)
  - one-to-one, **one-to-many**, separations, etc.

```
# if matchcoord is a single position
d2d = matchcoord.separation(catalogcoord)
catmask = d2d < 1*u.deg

# if matchcoord is a list of positions
idxmatch, idxcatalog, d2d, d3d =
   catalogcoord.search_around_sky(matchcoord, 1*u.deg)
```

# Tables

astropy
A Community Python Library for Astronomy

- Tables
  - Read FITS, ASCII, and more
  - Nice interface, similar to numpy ndarray/recarray
  - Fast, powerful, easy to use, well documented
  - QTable: seamless support for units

```
>>> import astropy.table as tab
>>> Table = tab.Table
>>> data = Table.read('mycatalogue.fits')
>>> print(data)  # print abridged table to screen
>>> data  # even nicer in IPython notebook
```

# Csomology

astropy

A Community Python Library for Astronomy

- Cosmology

```
>>> from astropy.cosmology import WMAP9 as cosmo
>>> cosmo.H(0)
<Quantity 69.32 km / (Mpc s)>
>>> cosmo.comoving_distance([0.5, 1.0, 1.5])
<Quantity [ 1916.0694236 , 3363.07064333, 4451.74756242] Mpc>

>>> from astropy.cosmology import FlatLambdaCDM
>>> cosmo = FlatLambdaCDM(H0=70, Om0=0.3)
>>> cosmo
FlatLambdaCDM(H0=70 km / (Mpc s), Om0=0.3, Tcmb0=2.725 K,
   Neff=3.04, m_nu=[ 0.  0.  0.] eV)
```

- note that many variables here are `Quantities`, they have units!

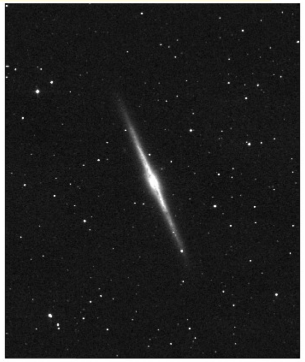
# Handling FITS files – astropy.io.fits

- FITS – file format for storing imaging and table data
  - very common in astronomy, but can be generally used
  - self describing, metadata, efficient, standardised
- Read, write and manipulate all aspects of FITS files
  - extensions
  - headers
  - images
  - tables (but typically use astropy.table)
- Low-level interface for details
- High-level functions for quick and easy use

# Reading FITS images

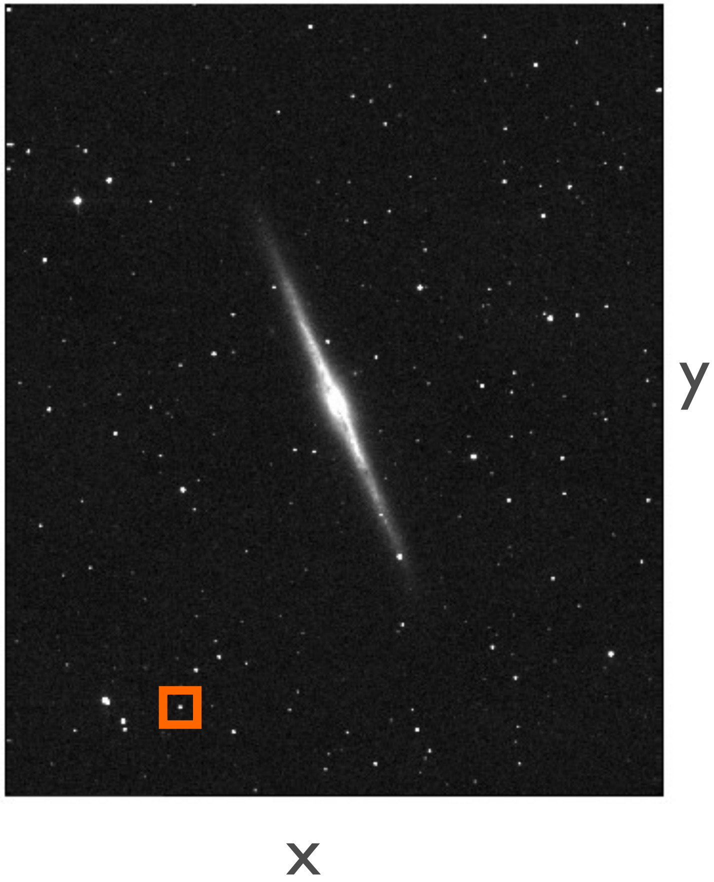
Very useful: `fits.info()`

```
>>> from astropy.io import fits
>>> imgname = 'data/2MASS_NGC_0891_K.fits'
>>> img = fits.getdata(imgname)
>>> img
array([[   0.        ,    0.        ,    0.        , ..., -999.00860596,
        -999.00860596, -999.00860596],
       [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
        -999.00860596, -999.00860596],
       [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
        -999.00860596, -999.00860596],
       ...,
       [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
        -999.00860596, -999.00860596],
       [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
        -999.00860596, -999.00860596],
       [-999.00860596, -999.00860596, -999.00860596, ..., -999.00860596,
        -999.00860596, -999.00860596]], dtype=float32)
>>> img.mean()
-8.6610549999999993
>>> img[img > -99].mean()
0.83546290095423026
>>> np.median(img)
0.078269213438034058
```



# Reading FITS images

```
>>> x = 348; y = 97
>>> delta = 5
>>> print img[y-delta:y+delta+1,
...               x-delta:x+delta+1].astype(np.int)
[[  1   1   1   1   1   0   0   0   1   0  -2]
 [  2   2   4   6   7   7   4   3   1   0  -1]
 [  1   4  11  24  40  40  21   7   2   0   0]
 [  1   6  23  62 110 107  50  13   2   0   0]
 [  2   7  33  91 158 148  68  15   3   0   0]
 [  3   7  27  74 123 115  53  12   2   0   0]
 [  2   4  12  32  54  51  24   5   1   0   0]
 [  1   1   2   7  12  12   5   0   0   0   0]
 [  0   0   0   1   2   2   1   0   0   1   0]
 [  0   0   0   1   0   0   0   0   0   0   0]
 [ -1   0   1   0   0   0   0   0   0   0   0]]
```




- row = y = first index
- column = x = second index
- numbering runs as normal (e.g. in ds9) BUT zero indexed!

# Writing FITS images

```
>>> newimg = sqrt((sky+img)/gain + rd_noise**2) * gain
>>> newimg[(sky+img) < 0.0] = 1e10

>>> hdr = h.copy()  # copy header from original image
>>> hdr.add_comment('Calculated noise image')

>>> filename = 'sigma.fits'

>>> pyfits.writeto(filename, newimg, hdr)  # create new file

>>> pyfits.append(imgname, newimg, hdr) # add a new FITS extension

>>> pyfits.update(filename, newimg, hdr, ext) # update a file

# specifying a header is optional,
# if omitted automatically adds minimum header
```

# Reading FITS headers

```
>>> h = pyfits.getheader(imgname)
>>> print h
SIMPLE  =                    T
BITPIX  =                  -32
NAXIS   =                    2
NAXIS1  =                 1000
NAXIS2  =                 1200
BLOCKED =                    T / TAPE MAY BE BLOCKED IN MULTIPLES OF 2880
EXTEND  =                    T / TAPE MAY HAVE STANDARD FITS EXTENSIONS
BSCALE  =                   1.
BZERO   =                   0.
ORIGIN  = '2MASS   '           / 2MASS Survey Camera
CTYPE1  = 'RA---SIN'
CTYPE2  = 'DEC--SIN'
CRPIX1  =                500.5
CRPIX2  =                600.5
CRVAL1  =          35.63922882
CRVAL2  =          42.34915161
CDELT1  =     -0.0002777777845
CDELT2  =      0.0002777777845
CROTA2  =                   0.
EQUINOX =                2000.
KMAGZP  =          20.07760048 /   V3 Photometric zero point calibration
COMMENTC= 'CAL updated by T.H. Jarrett, IPAC/Caltech'
SIGMA   =          1.059334397 /   Background Residual RMS noise (dn)
COMMENT1= '2MASS mosaic image'
COMMENT2= 'created by T.H. Jarrett, IPAC/Caltech'
>>> h['KMAGZP']
20.077600480000001
# Use h.items() to iterate through all header entries
```

# Low level usage

```
>>> f = pyfits.open(tblname)
>>> f.info()
Filename: data/N891PNdata.fits
No.    Name          Type       Cards   Dimensions   Format
0    PRIMARY     PrimaryHDU       4   ()             uint8
1                BinTableHDU     52   223R x 22C     [E, E, E, E, E,
   E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E, E]

>>> table = f[1]   # data extension number 1 (can also use names)

>>> d = f[1].data  # data, same as returned by pyfits.getdata()
>>> h = f[1].header  # header, same returned by pyfits.getheader()

>>> # make any changes
>>> f.writeto(othertblname)  # writes (with changes) to a new file

>>> f = pyfits.open(tblname, mode='update')  # to change same file
>>> # make any changes
>>> f.flush()  # writes changes back to file
>>> f.close()  # writes changes and closes file
```

# Memory mapping

- Useful if you only need to access a small region of a large *image*
- Only reads elements from disk as accessed, not whole image

```
>>> p = pyfits.open('gal.fits')
>>> d = p[0].data     # wait… data now in memory as a numpy array
>>> p = pyfits.open('gal.fits', memmap=True)
>>> d = p[0].data              # data still on disk, not in memory
>>> type(d)
<class 'np.core.memmap.memmap'>
>>> x = d[10:12, 10:12]     # only small amount of data in memory
>>> x
memmap([[  2.92147326,   0.73809952],
       [-16.27580261, -13.62474442]], dtype=float32)
```

- Only works for files up to ~2Gb (due to limit on Python object size)

# NDData and CCDData

- NDData
  - numpy arrays with support for meta data, uncertainties, etc.
- CCDData
  - class for handling images, understands WCS

```
>>> from astropy.nddata import CCDData

>>> ccd = = CCDData.read('test_file.fits’, unit=‘adu’)
>>> ccd.mask = ccd.data < -99
>>> ccd.uncertainty = np.ma.sqrt(np.ma.abs(ccd.data))
>>> ccd.write(‘test_file.fits’)
```

# astropy

astropy
A Community Python Library for Astronomy

- Affiliated packages
  - ccdproc – data reduction
  - photutils – photometry (also see SEP)
  - specutils – spectroscopy
  - astroplan – observation planning
  - astroML – machine learning methods
  - … and many more

# PyRAF

**IRAF: astronomical image reduction environment**

- Several decades of history and development
- Still quite widely used tool, but rapidly fading out
- Reduction packages for new instruments are usually written as standalone software (generally utilising astropy)
- If you need it, your supervisor will tell you (but even then, maybe not)
- **PyRAF – Python interface to IRAF**
- STScI provides installation package:

  https://astroconda.readthedocs.io/en/latest/

## PHYS4038/MLiS and AS1/MPAGS

# Scientific Programming in python™

mpags-python.github.io

Steven Bamford

University of Nottingham
UK | CHINA | MALAYSIA

An introduction to scientific programming with

python™

**Session 7.2**:

Python for theorists

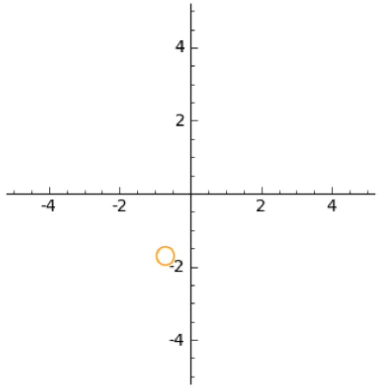
# Python for theorists

- http://www.sagemath.org/
- Python-based mathematics software
  - replacement for Maple, Mathematica
  - runs as a web application
  - local and online (cocalc.com)
  - private and collaborative workbooks

```
def r2(zz0):
    zz = complex(zz0) # coerce symbolic expression -> complex number
    return zz.real, zz.imag

def isoshow_act(ff,start,times=7,radius=1/4): #apply ff to start times times
    if times>10:
        times=10
    pts = [start]
    for k in range(times):
        start = ff(start)
        pts.append(start)
    return [circle(r2(pts[i]), radius, hue=i/10)
            for i in range(0,times)]

ll=isoshow_act(compose(-f1,f3),-1,times=4)
a = animate(ll,xmin=-5,ymin=-5,xmax=5,ymax=5,figsize=[4,4])
a.show()
```




**Examples:**

```
var('z')
f1(z)=-z+I             # recall that i, the sqrt of -1, is denoted by I in Sage
print f1(5-2*I)
f2(z)=conjugate(z)     # this is the reflection w.r.t. the x-axis
print f2(I), " ", f2(1)
f3(z)=(cos(pi/4)+sin(pi/4)*I)*z # rotation by pi/4
print f3(1), " ", f3(I-3), f2(f3(I-3))
```

```
3*I - 5
-I   1
(1/2*I + 1/2)*sqrt(2)   -(I + 2)*sqrt(2) (I - 2)*sqrt(2)
```

# Python for theorists

- SymPy: http://sympy.org/
- Python library for symbolic mathematics
- Comprehensive documentation
  - with built-in live Sympy shell
  - http://docs.sympy.org
- Use online
  - http://live.sympy.org

# SymPy – numbers

- Arbitrary precision
- Rationals and symbols for special constants and irrationals

```
>>> from sympy import *
>>> a = Rational(1,2)  # create a Rational number
>>> a, a*2, a**2
(1/2, 1, 1/4)
>>> sqrt(8)            # propagates surds
2*2**(1/2)
>>> (exp(pi))**2       # special constants
exp(2*pi)
>>> exp(pi).evalf()    # explicitly request float representation
23.1406926327793
>>> oo > 99999         # infinity
True
```

Thanks to Fabian Pedregosa

http://scipy-lectures.github.com/advanced/sympy.html

# SymPy – algebra

- Can define variables to be treated as symbols
- Expressions can be manipulated algebraically

```
>>> x = Symbol('x')
>>> y = Symbol('y')

>>> x+y+x-y
2*x
>>> (x+y)**2
(x + y)**2

>>> expand((x+y)**3)
3*x*y**2 + 3*y*x**2 + x**3 + y**3

>>> simplify((x+x*y)/x)
1 + y
```

```
# define multiple symbols
>>> x, y, z = symbols('x,y,z')

# useful shortcut
>>> f = simpify('(x+y)**2')

# latex output!
>>> print latex(exp(x**2/2))
e^{\frac{1}{2} x^{2}}
```

# SymPy – calculus

- Limits, derivatives, Taylor expansions and integrals

```
>>> limit(sin(x)/x, x, 0)

>>> diff(tan(x), x)
1 + tan(x)**2

>>> limit((tan(x+y)-tan(x))/y, y, 0)          # check using limit!
1 + tan(x)**2

>>> diff(sin(2*x), x, 3)                   # higher order derivatives
-8*cos(2*x)

>>> series(1/cos(x), x, pi/2, 5)          # around x=pi/2 to 5th order
-1/x - x/6 - 7*x**3/360 + O(x**5)
```

# SymPy – calculus

- Indefinite and definite integration

```
>>> integrate(sin(x), x)
-cos(x)
>>> integrate(log(x), x)
-x + x*log(x)

>>> integrate(exp(-x**2)*erf(x), x)   # including special functions
pi**(1/2)*erf(x)**2/4

>>> integrate(sin(x), (x, 0, pi/2))          # definite integral
1
>>> integrate(exp(-x**2), (x, -oo, oo))      # improper integral
pi**(1/2)
```

# SymPy – equation solving

- solve(f, x) returns the values of x which satisfy f(x) = 0
- f and x can be tuples → simultaneous equations
- Can also factorise polynomials

```
>>> solve(x**4 - 1, x)
[I, 1, -1, -I]
>>> solve(exp(x) + 1, x)
[pi*I]

>>> solve([x + 5*y - 2, -3*x + 6*y - 15], [x, y])
{y: 1, x: -3}

>>> f = x**4 - 3*x**2 + 1
>>> factor(f)
(1 + x - x**2)*(1 - x - x**2)
```

# SymPy – matrices

- Linear algebra

```
>>> m = Matrix([[1, 1, -1], [1, -1, 1], [-1, 1, 1]])

>>> m.inv()
[1/2, 1/2,   0]
[1/2,   0, 1/2]
[  0, 1/2, 1/2]

>>> P, D = m.diagonalize()
>>> D
[1, 0,  0]
[0, 2,  0]
[0, 0, -2]
>>> D ==  P.inv() * m * P
True
```

# SymPy – differential equations

- Can solve some ODEs

```
>>> g = f(x).diff(x, x) + f(x)

>>> dsolve(g, f(x))
f(x) == C1*cos(x) + C2*sin(x)


# sometimes a hint is helpful:

>>> dsolve(sin(x)*cos(f(x)) + cos(x)*sin(f(x))*f(x).diff(x), f(x),
   hint='separable')
-log(1 - sin(f(x))**2)/2 == C1 + log(1 - sin(x)**2)/2

>>> dsolve(x*f(x).diff(x) + f(x) - f(x)**2, f(x), hint='Bernoulli')
f(x) == 1/(x*(C1 + 1/x))
```

# SymPy – Physics module

- Quantum mechanics, classical mechanics, Gaussian optics and more

```
>>> from sympy import symbols, pi, diff
>>> from sympy.functions import sqrt, sin
>>> from sympy.physics.quantum.state import Wavefunction
>>> x, L = symbols('x,L', positive=True)
>>> n = symbols('n', integer=True)
>>> g = sqrt(2/L)*sin(n*pi*x/L)
>>> f = Wavefunction(g, (x, 0, L))
>>> f.norm
1
>>> f(L-1)
sqrt(2)*sin(pi*n*(L - 1)/L)/sqrt(L)
>>> f(0.85, n=1, L=1)
sqrt(2)*sin(0.85*pi)
```

But, also see QuTiP…

# SymPy – Physics module

- Units

```
>>> from sympy.physics.units import *

>>> 300*kilo*20*percent        # dimensionless units
60000

>>> milli*kilogram             # SI units
kg/1000
>>> gram
kg/1000
>>> joule
kg*m**2/s**2
```

- Astropy and others also provide units