Time and Coordinates

astropy.time

Python's built-in `datetime` package handles standard dates and times, e.g.,

```
In [1]: import datetime
```

```
In [2]: # January 4, 2020
        dt = datetime.datetime(2020, 1, 4)
```

```
In [3]: # January 4, 2020 10:30AM
        dt = datetime.datetime(2020, 1, 4, 10, 30)
```

but it doesn't support astronomical formats (e.g., Julian Date, Modified JD) or precise timing (e.g., pulsar timing)

astropy.time

The `astropy.time` subpackage provides support for representing date/time information with higher precision, and transforming between different formats (e.g., JD, MJD) and scales (e.g., TAI, UTC, solar system barycentric)

Key object: `Time`

```
In [4]: from astropy.time import Time
```

astropy.time

In [5]:
```
t = Time(58852.76, format='mjd', scale='tai')
```

astropy.time

value(s) *format* *scale*

In [5]:
```
t = Time(58852.76, format='mjd', scale='tai')
```

astropy.time

value(s) *format* *scale*

```
In [5]: t = Time(58852.76, format='mjd', scale='tai')

In [6]: t.jd

Out[6]: 2458853.26
```

astropy.time

value(s) *format* *scale*

```
In [5]: t = Time(58852.76, format='mjd', scale='tai')
```

```
In [6]: t.jd
```

Access other formats as attributes (returns a value)

```
Out[6]: 2458853.26
```

astropy.time

value(s) *format* *scale*

```
In [5]: t = Time(58852.76, format='mjd', scale='tai')

In [6]: t.jd
```

Access other formats as attributes (returns a value)

```
Out[6]: 2458853.26

In [7]: t.tcb

Out[7]: <Time object: scale='tcb' format='mjd' value=58852.76061606828>
```

astropy.time

value(s) *format* *scale*

```
In [5]: t = Time(58852.76, format='mjd', scale='tai')

In [6]: t.jd
```

Access other formats as attributes (returns a value)

```
Out[6]: 2458853.26

In [7]: t.tcb
```

Or, access other scales (returns a new Time object)

```
Out[7]: <Time object: scale='tcb' format='mjd' value=58852.76061606828>
```

astropy.time

value(s) *format* *scale*

```
In [5]: t = Time(58852.76, format='mjd', scale='tai')

In [6]: t.jd
```

Access other formats as attributes (returns a value)

```
Out[6]: 2458853.26

In [7]: t.tcb
```

Or, access other scales (returns a new Time object)

```
Out[7]: <Time object: scale='tcb' format='mjd' value=58852.76061606828>

In [8]: t.tcb.jd

Out[8]: 2458853.2606160683
```

astropy.time

value(s) *format* *scale*

```
In [5]: t = Time(58852.76, format='mjd', scale='tai')

In [6]: t.jd
Out[6]: 2458853.26

In [7]: t.tcb
Out[7]: <Time object: scale='tcb' format='mjd' value=58852.76061606828>

In [8]: t.tcb.jd
Out[8]: 2458853.2606160683
```

Access other formats as attributes (returns a value)

Or, access other scales (returns a new Time object)

String them together to get a new format in a new scale

astropy.time

value(s) *format* *scale*

```
In [5]: t = Time(58852.76, format='mjd', scale='tai')

In [6]: t.jd
Out[6]: 2458853.26

In [7]: t.tcb
Out[7]: <Time object: scale='tcb' format='mjd' value=58852.76061606828>

In [8]: t.tcb.jd
Out[8]: 2458853.2606160683

In [9]: t.datetime
Out[9]: datetime.datetime(2020, 1, 4, 18, 14, 24)
```

Access other formats as attributes (returns a value)

Or, access other scales (returns a new Time object)

String them together to get a new format in a new scale

astropy.coordinates

For representing and transforming astronomical coordinates and velocities

For example: you are given a set of equatorial (RA/Dec) coordinates in sexagesimal form:

```
00:48:26.4 +85:15:36
```

and you would like to transform these to Galactic coordinates (l, b) in decimal form

```
122.86494563, 22.3886423
```

astropy.coordinates

For representing and transforming astronomical coordinates and velocities

Key object: `SkyCoord`

In [1]:
```
from astropy.coordinates import SkyCoord
import astropy.units as u
```

In [2]:
```
c = SkyCoord(ra='00:48:26.4',
             dec='+85:15:36',
             unit=(u.hourangle, u.degree))
c
```

Out[2]:
```
<SkyCoord (ICRS): (ra, dec) in deg
    (12.11, 85.26)>
```

In [3]:
```
c.galactic
```

Out[3]:
```
<SkyCoord (Galactic): (l, b) in deg
    (122.86494563, 22.3886423)>
```

astropy.coordinates

Key object: `SkyCoord`

High-level object for representing, transforming, re-formatting, interacting with sky coordinates

```
In [4]: SkyCoord(ra=12.11*u.deg, dec=85.26*u.deg,
                 distance=147*u.pc)

Out[4]: <SkyCoord (ICRS): (ra, dec, distance) in (deg, deg, pc)
            (12.11, 85.26, 147.)>
```

astropy.coordinates

Key object: `SkyCoord`

High-level object for representing, transforming, re-formatting, interacting with sky coordinates

```
In [4]: SkyCoord(ra=12.11*u.deg, dec=85.26*u.deg,
                 distance=147*u.pc)
```

```
Out[4]: <SkyCoord (ICRS): (ra, dec, distance) in (deg, deg, pc)
            (12.11, 85.26, 147.)>
```

Also supports distance and velocity data

```
In [5]: SkyCoord(ra=12.11*u.deg, dec=85.26*u.deg,
                 distance=147*u.pc,
                 pm_ra_cosdec=10*u.mas/u.yr, pm_dec=21*u.mas/u.yr,
                 radial_velocity=-92.4*u.km/u.s)
```

```
Out[5]: <SkyCoord (ICRS): (ra, dec, distance) in (deg, deg, pc)
            (12.11, 85.26, 147.)
         (pm_ra_cosdec, pm_dec, radial_velocity) in (mas / yr, mas / yr, km / s)
            (10., 21., -92.4)>
```

astropy.coordinates

Key object: `SkyCoord`

Also: array-valued coordinate data!

```
In [6]: SkyCoord(ra=[12.11, 15.45, 82.2]*u.deg,
                 dec=[85.26, 23.4, -11.34]*u.deg)

Out[6]: <SkyCoord (ICRS): (ra, dec) in deg
            [(12.11,  85.26), (15.45,  23.4 ), (82.2 , -11.34)]>
```

astropy.coordinates

Supports many coordinate ***frames***

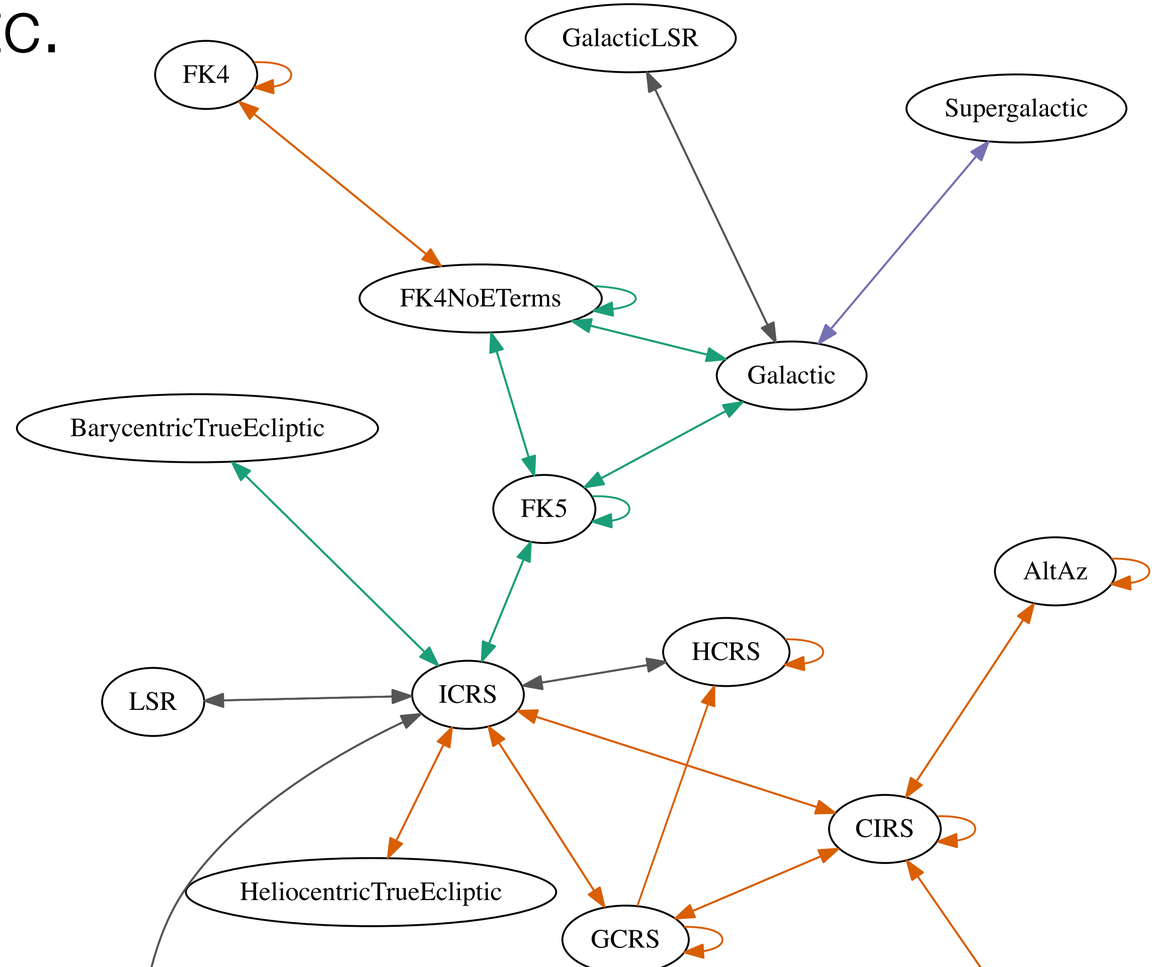
e.g., ICRS (J2000 equatorial), Galactic, FK5, AltAz (Horizontal), etc.

astropy.coordinates

Supports many coordinate ***frames***

e.g., ICRS (J2000 equatorial), Galactic, FK5, AltAz (Horizontal), etc.

The frame transform graph:

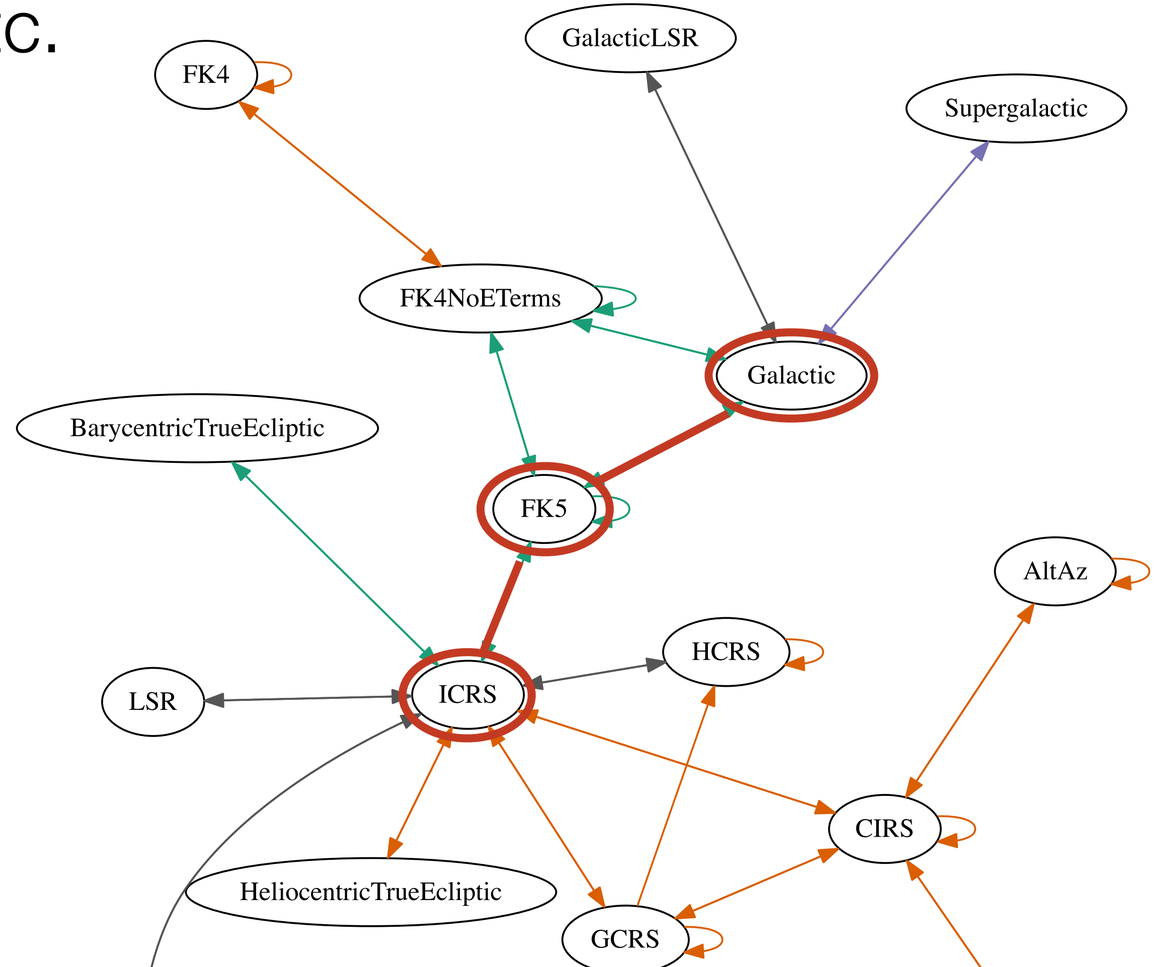


astropy.coordinates

Supports many coordinate ***frames***

e.g., ICRS (J2000 equatorial), Galactic, FK5, AltAz (Horizontal), etc.

The frame transform graph:



astropy.coordinates

Each frame has its own *class* implemented in astropy.coordinates:

In [7]:
```
c = SkyCoord(ra=[12.11, 15.45, 82.2]*u.deg,
             dec=[85.26, 23.4, -11.34]*u.deg)

from astropy.coordinates import Galactic
c.transform_to(Galactic())
```

Out[7]:
```
<SkyCoord (Galactic): (l, b) in deg
    [(122.86494563,  22.3886423 ), (126.00948245, -39.40982015),
     (214.00989281, -23.4238516 )]>
```

astropy.coordinates

Each frame has its own *class* implemented in astropy.coordinates:

In [7]:
```
c = SkyCoord(ra=[12.11, 15.45, 82.2]*u.deg,
             dec=[85.26, 23.4, -11.34]*u.deg)

from astropy.coordinates import Galactic
c.transform_to(Galactic())
```

Out[7]:
```
<SkyCoord (Galactic): (l, b) in deg
    [(122.86494563,  22.3886423 ), (126.00948245, -39.40982015),
     (214.00989281, -23.4238516 )]>
```

In [8]:
```
c.transform_to('galactic')
c.galactic
```

can also use a string alias or shorthand class name

Out[8]:
```
<SkyCoord (Galactic): (l, b) in deg
    [(122.86494563,  22.3886423 ), (126.00948245, -39.40982015),
     (214.00989281, -23.4238516 )]>
```

array-like times/coordinates

The core objects in both `astropy.time` and `astropy.coordinates` accept scalar *or* array-valued data! Most operations (transformations, re-representation) will be much faster with array-valued objects vs. looping over scalar-valued objects!

array-like times/coordinates

The core objects in both `astropy.time` and `astropy.coordinates` accept scalar *or* array-valued data! Most operations (transformations, re-representation) will be much faster with array-valued objects vs. looping over scalar-valued objects!

In [9]:
```
ra = [1., 2., 5., 10.] * u.deg
dec = [-10., -20., -30., -40.] * u.deg
SkyCoord(ra=ra, dec=dec).transform_to('galactic')
```

Out[9]:
```
<SkyCoord (Galactic): (l, b) in deg
    [( 87.54484429, -69.54375295), ( 66.26011125, -77.76707822),
     (  8.49560981, -82.52603149), (312.64746887, -76.91171264)]>
```

In [10]:
```
# NOT:
for i in range(len(ra)):
    SkyCoord(ra=ra[i], dec=dec[i]).transform_to('galactic')
```

array-like times/coordinates

The core objects in both `astropy.time` and `astropy.coordinates` accept scalar *or* array-valued data! Most operations (transformations, re-representation) will be much faster with array-valued objects vs. looping over scalar-valued objects!

In [9]:
```
ra = [1., 2., 5., 10.] * u.deg
dec = [-10., -20., -30., -40.] * u.deg
SkyCoord(ra=ra, dec=dec).transform_to('galactic')
```

Out[9]:
```
<SkyCoord (Galactic): (l, b) in deg
    [( 87.54484429, -69.54375295), ( 66.26011125, -77.76707822),
     (  8.49560981, -82.52603149), (312.64746887, -76.91171264)]>
```

In [10]:
```
# NOT:
for i in range(len(ra)):
    SkyCoord(ra=ra[i], dec=dec[i]).transform_to('galactic')
```

This will generally be slower!

Tutorial

Open up `astropy_coordinates.ipynb` and dive in!