PyEphem

Page Contents

Previous Page
Welcome!

Next Page
The PyEphem Tutorial

PyEphem Quick Reference

Those experienced with both Python and astronomy should be able to start using PyEphem using only the notes and examples shown below! There are two ways to begin using PyEphem in a Python program. One is to import the module by name, and then to prefix everything you want to use from the module with the qualifier ephem; this is the way the code snippets below are written, which hopefully makes it clear which variables are coming from PyEphem itself and which are being created in the course of each example.

```
>>> import ephem
>>> m = ephem.Mars('1970')
>>> print ephem.constellation(m)
('Aqr', 'Aquarius')
```

But to avoid typing the module name over and over again, you can tell Python to import everything from the module right into your namespace, where you can then use them without further qualification:

```
>>> from ephem import *
>>> m = Mars('1970')
```

```
>>> print constellation(m)
('Aqr', 'Aquarius')
```

To understand each of the following examples, first read the source code snippet carefully, and only then dive into the explanations beneath it.

Bodies

```
>>> m = ephem.Mars()
>>> m.name
'Mars'
>>> a = ephem.star('Arcturus')
>>> a.name
'Arcturus'
```

- The Sun, Moon, planets, and major planet moons each have their own class.
- PyEphem includes a modest catalog of famous bright stars.
- Body instances know their name (which you can set to whatever you want).

```
>>> m = ephem.Mars('2003/8/27')
>>> print m.name, m.elong, m.size
Mars -173:00:34.2 25.1121063232
```

- Extra arguments when you create a Body are used to perform an initial `compute()` (see the next section).

body.compute(date)

```
>>> j = ephem.Jupiter()
>>> j.compute('1986/2/8')
>>> print j.ra, j.dec
21:57:50.46 -13:17:37.2
>>> j.compute('1986/2/9', epoch='1950')
>>> print j.a_ra, j.a_dec
21:56:50.83 -13:22:54.3
```

- Computes the position of the body.
- The date if omitted defaults to `now()`.
- The epoch if omitted defaults to `'2000'`.
- Date and epoch arguments can be anything acceptable to `Date()`.
- Sets the following body attributes:

  `a_ra` — Astrometric geocentric right ascension for the epoch specified
  `a_dec` — Astrometric geocentric declination for the epoch specified
  `g_ra` and `ra` — Apparent geocentric right ascension for the epoch-of-date
  `g_dec` and `dec` — Apparent geocentric declination for the epoch-of-date
  `elong` — Elongation (angle to sun)
  `mag` — Magnitude
  `size` — Size (diameter in arcseconds)
  `radius` — Size (radius as an angle)

- On Solar System bodies, also sets:

`hlon` — Heliocentric longitude (see next paragraph)
`hlat` — Heliocentric latitude (see next paragraph)
`sun_distance` — Distance to Sun (AU)
`earth_distance` — Distance to Earth (AU)
`phase` — Percent of surface illuminated

Note that `hlon` and `hlat` on the Sun object, which would normally have no meaning since those angles are measured from the Sun's point of view, instead give the heliocentric longitude and latitude of Earth.

- On planetary moons, also sets:

  Position of moon relative to planet
  (measured in planet radii)
  `x` — offset +east or –west
  `y` — offset +south or –north
  `z` — offset +front or –behind
  Whether the moon is visible...
  `earth_visible` — from the Earth
  `sun_visible` — from the Sun

- On artificial satellites, also sets:

  Geographic point beneath satellite:
  `sublat` — Latitude (+N)
  `sublong` — Longitude (+E)

  `elevation` — Height above sea level (m)
  `range` — Distance from observer to satellite (m)
  `range_velocity` — Range rate of change (m/s)
  `eclipsed` — Whether satellite is in Earth's shadow

- On `Moon` bodies, also sets:

  Current libration:
  `libration_lat` — in Latitude
  `libration_long` — in Longitude

  `colong` — Selenographic colongiude
  `moon_phase` — Percent of surface illuminated
  `subsolar_lat` — Lunar latitude that the Sun is standing above

- On `Jupiter` bodies, also determines the longitude of the central meridian facing Earth, both in System I (which measures rotation at the Jovial equator) and System II (which measures rotation at temperate latitudes).

  `cmlI` — Central meridian longitude in System I
  `cmlII` — Central meridian longitude in System II

- On `Saturn` bodies, also sets the tilt of the rings, with southward tilt being positive, and northward, negative:

  `earth_tilt` — Tilt towards Earth
  `sun_tilt` — Tilt towards Sun

body.compute(observer)

```
>>> gatech = ephem.Observer()
>>> gatech.lon = '-84.39733'
```

```
>>> gatech.lat = '33.775867'
>>> gatech.elevation = 320
>>> gatech.date = '1984/5/30 16:22:56'
>>> v = ephem.Venus(gatech)
>>> print v.alt, v.az
72:19:44.8 134:14:25.3
```

- Computes the position of the Body.
- Uses the `date` of the observer.
- Uses the `epoch` of the observer.
- Sets all of the Body attributes listed in the previous section (but `ra` and `dec` will get different values; see below).
- Also computes where the body appears in the sky (or below the horizon) for the observer, and sets four more Body attributes:

  Apparent topocentric position
  `ra` — Right ascension
  `dec` — Declination

  Apparent position relative to horizon
  `az` — Azimuth east of north
  `alt` — Altitude above horizon

- These apparent positions include an adjustment to simulate atmospheric refraction for the observer's `temp` and `presure`; set the observer's `pressure` to zero to ignore refraction.
- For earth satellite objects, the astrometric coordinates are topocentric instead of geocentric, since there is little point in figuring out where the satellite would appear on a J2000 (or whatever epoch you are using) star chart for an observer sitting at the center of the earth.

catalog format

```
>>> line = "C/2002 Y1 (Juels-Holvorcem),e,103.7816,166.2194,128.8232,242.5695,0.0002
>>> yh = ephem.readdb(line)
>>> yh.compute('2007/10/1')
>>> print yh.earth_distance
14.8046731949
>>> print yh.mag
23.96
```

- Bodies can be imported and exported in the popular XEphem format.
- When you deal with asteroids and comets, whose orbital parameters are subject to frequent revision, you will usually find yourself downloading an XEphem file and reading its contents.
- To interpret a line in XEphem format, call the `readdb()` function:

```
halley = ephem.readdb(line)
```

- To export a body in XEphem format, call the `writedb()` method of the body itself:

```
    print halley.writedb()
```

```
>>> line1 = "ISS (ZARYA)"
>>> line2 = "1 25544U 98067A   03097.78853147  .00021906  00000-0  28403-3 0  8652"
>>> line3 = "2 25544  51.6361  13.7980 0004256  35.6671  59.2566 15.58778559250029"
>>> iss = ephem.readtle(line1, line2, line3)
>>> iss.compute('2003/3/23')
>>> print iss.sublong, iss.sublat
-76:24:18.3 13:05:31.1
```

- Satellite elements often come packaged in a format called TLE, that has the satellite name on one line and the elements on the following two lines.
- Call the `readtle()` function to turn a TLE entry into a PyEphem `Body`.

bodies with orbital elements

- When you load minor objects like comets and asteroids, the resulting object specifies the *orbital elements* that allow XEphem to predict its position.
- These orbital elements are available for you to inspect and change.
- If you lack a catalog from which to load an object, you can start by creating a raw body of one of the following types and filling in its elements.
- Element attribute names start with underscores to distinguish them from the normal `Body` attributes that are set as the result of calling `compute()`.
- Each `FixedBody` has only three necessary elements:

  `_ra`, `_dec` — Position
  `_epoch` — The epoch of the position

  The other `FixedBody` elements store trivia about its appearance:

  `_class` — One-character string classification
  `_spect` — Two-character string for the spectral code
  `_ratio` — Ratio between the major and minor diameters
  `_pa` — the angle at which the major axis lies in the sky, measured east of north (°)

- `EllipticalBody` elements:

  `_inc` — Inclination (°)
  `_Om` — Longitude of ascending node (°)
  `_om` — Argument of perihelion (°)
  `_a` — Mean distance from sun (AU)
  `_M` — Mean anomaly from the perihelion (°)
  `_epoch_M` — Date for measurement `_M`
  `_size` — Angular size (arcseconds at 1 AU)
  `_e` — Eccentricity
  `_epoch` — Epoch for `_inc`, `_Om`, and `_om`
  `_H`, `_G` — Parameters for the H/G magnitude model
  `_g`, `_k` — Parameters for the g/k magnitude model

- `HyperbolicBody` elements:

  `_epoch` — Equinox year for `_inc`, `_Om`, and `_om`
  `_epoch_p` — Epoch of perihelion
  `_inc` — Inclination (°)

_Om — Longitude of ascending node (°)
_om — Argument of perihelion (°)
_e — Eccentricity
_q — Perihelion distance (AU)
_g, _k — Magnitude model coefficients
_size — Angular size in arcseconds at 1 AU

- ParabolicBody elements:

  _epoch — Epoch for _inc, _Om, and _om
  _epoch_p — Epoch of perihelion
  _inc — Inclination (°)
  _Om — Longitude of ascending node (°)
  _om — Argument of perihelion (°)
  _q — Perihelion distance (AU)
  _g, _k — Magnitude model coefficients
  _size — Angular size in arcseconds at 1 AU

- EarthSatellite elements:

  _epoch — Reference epoch
  _n — Mean motion, in revolutions per day
  _inc — Inclination (°)
  _raan — Right Ascension of ascending node (°)
  _e — Eccentricity
  _ap — Argument of perigee at epoch (°)
  _M — Mean anomaly from perigee at epoch (°)
  _decay — Orbit decay rate in revolutions per day, per day
  _drag — Object drag coefficient in per earth radii
  _orbit — Integer orbit number of epoch

Other Functions

```
>>> m = ephem.Moon('1980/6/1')
>>> print ephem.constellation(m)
('Sgr', 'Sagittarius')
```

- The `constellation()` function returns a tuple containing the abbreviated name and full name of the constellation in which its argument lies.
- You can either pass a Body whose position is computed, or a tuple `(ra, dec)` of coordinates — in which case epoch 2000 is assumed unless you also pass an epoch= keyword argument specifying another value.

```
>>> print ephem.delta_t('1980')
50.54
```

- The `delta_t()` function returns the difference, in seconds, on the given date between Terrestrial Time and Universal Time.
- Takes a `Date` or `Observer` argument.
- Without an argument, uses `now()`.

```
>>> ephem.julian_date('2000/1/1')
2451544.5
```

- The `julian_date()` function returns the official Julian Date of the given day and time.
- Takes a `Date` or `Observer` argument.

- Without an argument, uses `now()`.

```
>>> ra, dec = '7:16:00', '-6:17:00'
>>> print ephem.uranometria(ra, dec)
V2 - P274
>>> print ephem.uranometria2000(ra, dec)
V2 - P135
>>> print ephem.millennium_atlas(ra, dec)
V1 - P273
```

- Take an `ra` and `dec` as arguments.
- Return the volume and page on which that coordinate lies in the given star atlas:

  Uranometria by Johannes Bayer.
  Uranometria 2000.0 edited by Wil Tirion.
  Millennium Star Atlas by Roger W. Sinnott and Michael A. C. Perryman.

```
>>> m1 = ephem.Moon('1970/1/16')
>>> m2 = ephem.Moon('1970/1/17')
>>> s = ephem.separation(m1, m2)
>>> print "In one day the Moon moved", s
In one day the Moon moved 12:33:28.5
```

- The `separation()` function returns the angle that separates two positions on a sphere.
- Each argument can be either a Body, in which case its `ra` and `dec` are used, or a tuple `(lon, lat)` giving a pair of spherical coordinates where `lon` measures angle around the sphere's equator and `lat` measures the angle above or below its equator.

Coordinate Conversion

```
>>> np = Equatorial('0', '90', epoch='2000')
>>> g = Galactic(np)
>>> print g.lon, g.lat
122:55:54.9 27:07:41.7
```

- There are three coordinate classes, which each have three properties:

  `Equatorial`
  `ra` — right ascension
  `dec` — declination
  `epoch` — epoch of the coordinate
  `Ecliptic`
  `lon` — ecliptic longitude (+E)
  `lat` — ecliptic latitude (+N)
  `epoch` — epoch of the coordinate
  `Galactic`
  `lon` — galactic longitude (+E)
  `lat` — galactic latitude (+N)
  `epoch` — epoch of the coordinate

- When creating a new coordinate, you can pass either a body, or another coordinate, or a pair of raw angles (always place the longitude or right ascension first).
- When creating a coordinate, you can optionally pass an epoch= keyword specifying the epoch for the coordinate system. Otherwise the epoch is copied from the body or other

coordinate being used, or J2000 is used as the default.

- See the Coordinate Transformations document for more details.

Observers

```
>>> lowell = ephem.Observer()
>>> lowell.lon = '-111:32.1'
>>> lowell.lat = '35:05.8'
>>> lowell.elevation = 2198
>>> lowell.date = '1986/3/13'
>>> j = ephem.Jupiter()
>>> j.compute(lowell)
>>> print j.alt, j.az
0:57:44.7 256:41:01.3
```

- Describes a position on Earth's surface.
- Pass to the `compute()` method of a `Body`.
- These are the attributes you can set:

  `date` — Date and time
  `epoch` — Epoch for astrometric RA/dec
  `lat` — Latitude (+N)
  `lon` — Longitude (+E)
  `elevation` — Elevation (m)
  `temp` — Temperature (°C)
  `pressure` — Atmospheric pressure (mBar)

- The `date` defaults to `now()`.
- The `epoch` defaults to `'2000'`.
- The `temp` defaults to 25°C.
- The `pressure` defaults to 1010mBar.
- Other attributes default to zero.

```
>>> lowell.compute_pressure()
>>> lowell.pressure
775.6025138640499
```

- Computes the pressure at the observer's current elevation, using the International Standard Atmosphere.

```
>>> boston = ephem.city('Boston')
>>> print boston.lat, boston.lon
42:21:30.4 -71:03:35.2
```

- XEphem includes a small database of world cities.
- Each call to `city()` returns a new `Observer`.
- Only latitude, longitude, and elevation are set.

transit, rising, setting

```
>>> sitka = ephem.Observer()
>>> sitka.date = '1999/6/27'
>>> sitka.lat = '57:10'
>>> sitka.lon = '-135:15'
>>> m = ephem.Mars()
>>> print sitka.next_transit(m)
1999/6/27 04:22:45
>>> print m.alt, m.az
21:18:33.6 180:00:00.0
>>> print sitka.next_rising(m, start='1999/6/28')
1999/6/28 23:28:25
>>> print m.alt, m.az
-0:00:05.8 111:10:41.6
```

- Eight `Observer` methods are available for finding rising, transit, and setting times:

```
previous_transit()
next_transit()

previous_antitransit()
next_antitransit()

previous_rising()
next_rising()

previous_setting()
next_setting()
```

- Each takes a Body argument, which can be any body except an `EarthSatellite` (for which the `next_pass()` method below should be used).
- Returns a `Date` value.
- Leaves the Body at its position on that date.
- The Observer itself is unchanged.
- Takes an optional `start=` argument giving the date and time from which the search for a rising, transit, or setting should commence.
- We define the meridian as the line running overhead from the celestial North pole to the South pole, and the anti-meridian as the other half of the same great circle; so the transit and anti-transit methods always succeed, whether the body crosses the horizon or not.
- But the rising and setting functions raise execptions if the body does not to cross the horizon; the exception hierarchy is:

```
ephem.CircumpolarError
 |
 +--- ephem.AlwaysUpError
 +--- ephem.NeverUpError
```

- Rising and setting are defined as the moments when the upper limb of the body touches the horizon (that is, when the body’s `alt` plus `radius` equals zero).
- Rising and setting are sensitive to atmospheric refraction at the horizon, and therefore to the observer’s `temp` and `pressure`; set the `pressure` to zero to turn off refraction.

- Rising and setting pay attention to the observer's `horizon` attribute; see the next section.

```
>>> line1 = "IRIDIUM 80 [+]"
>>> line2 = "1 25469U 98051C   09119.61415140 -.00000218  00000-0 -84793-4 0  4781"
>>> line3 = "2 25469  86.4029 183.4052 0002522  86.7221 273.4294 14.34215064557061"
>>> iridium_80 = ephem.readtle(line1, line2, line3)
>>> boston.date = '2009/5/1'
>>> info = boston.next_pass(iridium_80)
>>> print "Rise time: %s azimuth: %s" % (info[0], info[1])
Rise time: 2009/5/1 00:22:15 azimuth: 104:36:21.5
```

- The `next_pass()` method takes an `EarthSatellite` body and determines when it will next cross above the horizon.
- It returns a six-element tuple giving:

```
0  Rise time
1  Rise azimuth
2  Transit time
3  Transit altitude
4  Set time
5  Set azimuth
```

- Any of the tuple values can be None if that event was not found.

observer.horizon

```
>>> sun = ephem.Sun()
>>> greenwich = ephem.Observer()
>>> greenwich.lat = '51:28:38'
>>> print greenwich.horizon
0:00:00.0
>>> greenwich.date = '2007/10/1'
>>> r1 = greenwich.next_rising(sun)
>>> greenwich.pressure = 0
>>> greenwich.horizon = '-0:34'
>>> greenwich.date = '2007/10/1'
>>> r2 = greenwich.next_rising(sun)
>>> print 'Visual sunrise:', r1
Visual sunrise: 2007/10/1 05:59:29
>>> print 'Naval Observatory sunrise:', r2
Naval Observatory sunrise: 2007/10/1 05:59:50
```

- The `horizon` attribute defines your *horizon*, the altitude of the upper limb of a body at the moment you consider it to be rising and setting.
- The `horizon` defaults to zero degrees.
- The United States Naval Observatory, rather than computing refraction dynamically, uses a constant estimate of 34' of refraction at the horizon. So in the above example, rather than attempting to jury-rig values for `temp` and `pressure` that yield the magic 34', we turn off PyEphem refraction entirely and define the horizon itself as being at 34' altitude instead.
- To determine when a body will rise "high enough" above haze or obstacles, set `horizon` to a positive number of degrees.
- A negative value of `horizon` can be used when an observer is high off of the ground.

other Observer methods

```
>>> madrid = ephem.city('Madrid')
>>> madrid.date = '1978/10/3 11:32'
>>> print madrid.sidereal_time()
12:04:28.09
```

- Called without arguments.
- Returns the sidereal time for the observer's circumstances.

```
>>> ra, dec = madrid.radec_of(0, '90')
>>> print ra, dec
12:05:35.12 40:17:49.8
```

- Called like `radec_of(az, alt)`.
- Returns the apparent topocentric coordinates behind the given point in the sky.

Equinoxes & Solstices

```
>>> d1 = ephem.next_equinox('2000')
>>> print d1
2000/3/20 07:35:17
>>> d2 = ephem.next_solstice(d1)
>>> print d2
2000/6/21 01:47:51
>>> t = d2 - d1
>>> print "Spring lasted %.1f days" % t
Spring lasted 92.8 days
```

- Functions take a `Date` argument.
- Return a `Date`.
- Available functions:

```
previous_solstice()
next_solstice()

previous_equinox()
next_equinox()

previous_vernal_equinox()
next_vernal_equinox()
```

Phases of the Moon

```
>>> d1 = ephem.next_full_moon('1984')
>>> print d1
1984/1/18 14:05:10
>>> d2 = ephem.next_new_moon(d1)
>>> print d2
1984/2/1 23:46:25
```

- Functions take a Date argument.
- Return a Date.
- Available functions:

```
previous_new_moon()
next_new_moon()

previous_first_quarter_moon()
next_first_quarter_moon()

previous_full_moon()
next_full_moon()

previous_last_quarter_moon()
next_last_quarter_moon()
```

Angles

```
>>> a = ephem.degrees('180:00:00')
>>> print a
180:00:00.0
>>> a
3.141592653589793
>>> print "180° is %f radians" % a
180° is 3.141593 radians
>>> h = ephem.hours('1:00:00')
>>> deg = ephem.degrees(h)
>>> print "1h right ascension = %s°" % deg
1h right ascension = 15:00:00.0°
```

- Many Body and Observer attributes return their value as Angle objects.
- Most angles are measured in degrees.
- Only right ascension is measured in hours.
- You can also create angles yourself through two ephem functions:

  degrees() — return an Angle in degrees
  hours() — return an Angle in hours

- Each angle acts like a Python float.
- Angles always store floating-point radians.
- Only when printed, passed to str(), or formatted with '%s' do angles display themselves as degrees or hours.
- When setting an angle attribute in a body or observer, or creating angles yourself, you can provide either floating-point radians or a string with degrees or hours. The following angles are equivalent:

```
ephem.degrees(ephem.pi / 32)
ephem.degrees('5.625')
```

```
ephem.degrees('5:37.5')
ephem.degrees('5:37:30')
ephem.degrees('5:37:30.0')
ephem.hours('0.375')
ephem.hours('0:22.5')
ephem.hours('0:22:30')
ephem.hours('0:22:30.0')
```

- When doing math on angles, the results will often exceed the normal bounds for an angle. Therefore two attributes are provided for each angle:

  `norm` — returns angle normalized to [0, 2π).
  `znorm` — returns angle normalized to [-π, π).

- For more details see the Angle document.

Dates

```
>>> d = ephem.Date('1997/3/9 5:13')
>>> print d
1997/3/9 05:13:00
>>> d
35496.717361111114
>>> d.triple()
(1997, 3, 9.21736111111386)
>>> d.tuple()
(1997, 3, 9, 5, 13, 2.3748725652694702e-07)
>>> d + ephem.hour
35496.75902777778
>>> print ephem.date(d + ephem.hour)
1997/3/9 06:13:00
>>> print ephem.date(d + 1)
1997/3/10 05:13:00
```

- Dates are stored and returned as floats.
- Only when printed, passed to `str()`, or formatted with `'%s'` does a date express itself as a string giving the calendar day and time.
- Dates *always* use Universal Time, *never* your local time zone.
- Call `.triple()` to split a date into its year, month, and day.
- Call `.tuple()` to split a date into its year, month, day, hour, minute, and second.
- You can create `ephem.Date()` dates yourself in addition to those you will be returned by other objects.
- Call `ephem.now()` for the current date and time.
- When setting a date attribute in a body or observer, or creating angles yourself, you can provide either floating-point radians, a string, or a tuple. The following dates are equivalent:

```
ephem.Date(35497.7197916667)
ephem.Date('1997/3/10.2197916667')
ephem.Date('1997/3/10 05.275')
```

```
ephem.Date('1997/3/10 05:16.5')
ephem.Date('1997/3/10 05:16:30')
ephem.Date('1997/3/10 05:16:30.0')
ephem.Date((1997, 3, 10.2197916667))
ephem.Date((1997, 3, 10, 5, 16, 30.0))
```

- Dates store the number of days that have passed since noon Universal Time on the last day of 1899. By adding and subtracting whole numbers from dates, you can move several days into the past or future. If you want to move by smaller amounts, the following constants may be helpful:

```
ephem.hour
ephem.minute
ephem.second
```

- For more details see the Date document.

local time

```
>>> d = ephem.Date('1997/3/9 5:13')
>>> ephem.localtime(d)
datetime.datetime(1997, 3, 9, 0, 13, 0, 6)
>>> print ephem.localtime(d)
1997-03-09 00:13:00.000006
```

- The `localtime()` function converts a PyEphem date into a Python `datetime` object expressed in your local time zone.

Stars and Cities

```
>>> rigel = ephem.star('Rigel')
>>> print rigel._ra, rigel._dec
5:14:32.30 -8:12:06.0
```

- PyEphem provides a catalog of bright stars.
- Each call to `star()` returns a new `FixedBody` whose coordinates are those of the named star.

```
>>> stuttgart = ephem.city('Stuttgart')
>>> print stuttgart.lon, stuttgart.lat
9:10:50.8 48:46:37.6
```

- PyEphem knows 122 world cities.
- The `city()` function returns an `Observer` whose longitude, latitude, and elevation are those of the given city.

Other Constants

- PyEphem provides constants for the dates of a few major star-atlas epochs:

```
B1900
B1950
```

```
J2000
```

- PyEphem provides, for reference, the length of four distances, all in meters:

```
ephem.meters_per_au
ephem.earth_radius
ephem.moon_radius
ephem.sun_radius
```

- PyEphem provides the speed of light in meters per second:

```
ephem.c
```