Astral v1.4

Release 1.4

Simon Kennedy

Jul 20, 2017

Contents

Astral is a python module for calculating the times of various aspects of the sun and moon.

It calculates the following

Dawn The time in the morning when the sun is a specific number of degrees below the horizon.

Sunrise The time in the morning when the top of the sun breaks the horizon (asuming a location with no obscuring features.)

Solar Noon The time when the sun is at its highest point.

Solar Midnight The time when the sun is at its lowest point.

Sunset The time in the evening when the sun is about to disappear below the horizon (asuming a location with no obscuring features.)

Dusk The time in the evening when the sun is a specific number of degrees below the horizon.

Daylight The time when the sun is up i.e. between sunrise and sunset

Night The time between astronomical dusk of one day and astronomical dawn of the next

Twilight The time between dawn and sunrise or between sunset and dusk

The Golden Hour The time when the sun is between 4 degrees below the horizon and 6 degrees above.

The Blue Hour The time when the sun is between 6 and 4 degrees below the horizon.

Time At Elevation the time when the sun is at a specific elevation for either a rising or a setting sun.

Solar Azimuth The number of degrees clockwise from North at which the sun can be seen

Solar Elevation The number of degrees up from the horizon at which the sun can be seen

Rahukaalam “Rahukaalam or the period of Rahu is a certain amount of time every day that is considered inauspicious for any new venture according to Indian Vedic astrology”.

Moon Phase Calculates the phase of the moon for a specified date.

CHAPTER 1

Example

The following example demonstrates the functionality available in the module:

```
>>> import datetime
>>> from astral import Astral

>>> city_name = 'London'

>>> a = Astral()
>>> a.solar_depression = 'civil'

>>> city = a[city_name]

>>> print('Information for %s/%s\n' % (city_name, city.region))
Information for London/England

>>> timezone = city.timezone
>>> print('Timezone: %s' % timezone)
Timezone: Europe/London

>>> print('Latitude: %.02f; Longitude: %.02f\n' % \
>>>     (city.latitude, city.longitude))
Latitude: 51.60; Longitude: 0.08

>>> sun = city.sun(date=datetime.date(2009, 4, 22), local=True)
>>> print('Dawn:     %s' % str(sun['dawn']))
>>> print('Sunrise:  %s' % str(sun['sunrise']))
>>> print('Noon:     %s' % str(sun['noon']))
>>> print('Sunset:   %s' % str(sun['sunset']))
>>> print('Dusk:     %s' % str(sun['dusk']))
Dawn:     2009-04-22 05:12:56+01:00
Sunrise:  2009-04-22 05:49:36+01:00
Noon:     2009-04-22 12:58:48+01:00
Sunset:   2009-04-22 20:09:07+01:00
Dusk:     2009-04-22 20:45:52+01:00
```

If the location you want is not in the Astral geocoder then you need to construct a *Location* and fill in the values either with a tuple on initialization:

```
l = Location(('name', 'region',
              0.1, 1.2, 'timezone/name', 0))
l.sun()
```

or set the attributes after initialization:

```
l = Location()
l.name = 'name'
l.region = 'region'
l.latitude = 0.1
l.longitude = 1.2
l.timezone = 'US/Central'
l.elevation = 0
l.sun()
```

Note: *name* and *region* can be anything you like.

Access to the current geocoder can be made through the Astral class:

```
>>> a = Astral()
>>> geo = a.geocoder
>>> london = geo['London']
```

Timezone groups such as Europe can be accessed via attributes on the *AstralGeocoder*:

```
>>> geo = AstralGeocoder()
>>> europe = geo.europe.locations
>>> europe.sort()
>>> europe
['Aberdeen', 'Amsterdam', 'Andorra la Vella', 'Ankara', 'Athens', ...]
```

CHAPTER 2

Note on Localized Timezones

When creating a datetime object in a specific timezone do not use the *tzinfo* parameter to the datetime constructor. Instead please use the `localize()` method on the correct pytz timezone:

```
dt = datetime.datetime(2015, 1, 1, 9, 0, 0)
dt = pytz.timezone('Europe/London').localize(dt)
```

CHAPTER 3

License

This module is licensed under the terms of the Apache V2.0 license.

CHAPTER 4

Dependencies

Astral has one external Python dependency on ‘pytz’.

CHAPTER 5

Installation

To install Astral you should use the **pip** tool:

```
pip install astral
```

CHAPTER 6

Cities

The module includes location and time zone data for the following cities. The list includes all capital cities plus some from the UK. The list also includes the US state capitals and some other US cities.

Aberdeen, Abu Dhabi, Abu Dhabi, Abuja, Accra, Addis Ababa, Adelaide, Al Jubail, Albany, Albuquerque, Algiers, Amman, Amsterdam, Anchorage, Andorra la Vella, Ankara, Annapolis, Antananarivo, Apia, Ashgabat, Asmara, Astana, Asuncion, Athens, Atlanta, Augusta, Austin, Avarua, Baghdad, Baku, Baltimore, Bamako, Bandar Seri Begawan, Bangkok, Bangui, Banjul, Barrow-In-Furness, Basse-Terre, Basseterre, Baton Rouge, Beijing, Beirut, Belfast, Belgrade, Belmopan, Berlin, Bern, Billings, Birmingham, Birmingham, Bishkek, Bismarck, Bissau, Bloemfontein, Bogota, Boise, Bolton, Boston, Bradford, Brasilia, Bratislava, Brazzaville, Bridgeport, Bridgetown, Brisbane, Bristol, Brussels, Bucharest, Bucuresti, Budapest, Buenos Aires, Buffalo, Bujumbura, Burlington, Cairo, Canberra, Cape Town, Caracas, Cardiff, Carson City, Castries, Cayenne, Charleston, Charlotte, Charlotte Amalie, Cheyenne, Chicago, Chisinau, Cleveland, Columbia, Columbus, Conakry, Concord, Copenhagen, Cotonou, Crawley, Dakar, Dallas, Damascus, Dammam, Denver, Des Moines, Detroit, Dhaka, Dili, Djibouti, Dodoma, Doha, Douglas, Dover, Dublin, Dushanbe, Edinburgh, El Aaiun, Fargo, Fort-de-France, Frankfort, Freetown, Funafuti, Gaborone, George Town, Georgetown, Gibraltar, Glasgow, Greenwich, Guatemala, Hanoi, Harare, Harrisburg, Hartford, Havana, Helena, Helsinki, Hobart, Hong Kong, Honiara, Honolulu, Houston, Indianapolis, Islamabad, Jackson, Jacksonville, Jakarta, Jefferson City, Jerusalem, Juba, Jubail, Juneau, Kabul, Kampala, Kansas City, Kathmandu, Khartoum, Kiev, Kigali, Kingston, Kingston, Kingstown, Kinshasa, Koror, Kuala Lumpur, Kuwait, La Paz, Lansing, Las Vegas, Leeds, Leicester, Libreville, Lilongwe, Lima, Lincoln, Lisbon, Little Rock, Liverpool, Ljubljana, Lome, London, Los Angeles, Louisville, Luanda, Lusaka, Luxembourg, Macau, Madinah, Madison, Madrid, Majuro, Makkah, Malabo, Male, Mamoudzou, Managua, Manama, Manchester, Manchester, Manila, Maputo, Maseru, Masqat, Mbabane, Mecca, Medina, Memphis, Mexico, Miami, Milwaukee, Minneapolis, Minsk, Mogadishu, Monaco, Monrovia, Montevideo, Montgomery, Montpelier, Moroni, Moscow, Moskva, Mumbai, Muscat, N'Djamena, Nairobi, Nashville, Nassau, Naypyidaw, New Delhi, New Orleans, New York, Newark, Newcastle, Newcastle Upon Time, Ngerulmud, Niamey, Nicosia, Norwich, Nouakchott, Noumea, Nuku'alofa, Nuuk, Oklahoma City, Olympia, Omaha, Oranjestad, Orlando, Oslo, Ottawa, Ouagadougou, Oxford, P'yongyang, Pago Pago, Palikir, Panama, Papeete, Paramaribo, Paris, Perth, Philadelphia, Phnom Penh, Phoenix, Pierre, Plymouth, Podgorica, Port Louis, Port Moresby, Port of Spain, Port-Vila, Port-au-Prince, Portland, Portland, Porto-Novo, Portsmouth, Prague, Praia, Pretoria, Pristina, Providence, Quito, Rabat, Raleigh, Reading, Reykjavik, Richmond, Riga, Riyadh, Road Town, Rome, Roseau, Sacramento, Saint Helier, Saint Paul, Saint Pierre, Saipan, Salem, Salt Lake City, San Diego, San Francisco, San Jose, San Juan, San Marino, San Salvador, Sana, Sana'a, Santa Fe, Santiago, Santo Domingo, Sao Tome, Sarajevo, Seattle, Seoul, Sheffield, Singapore, Sioux Falls, Skopje, Sofia, Southampton, Springfield, Sri Jayawardenapura Kotte, St. George's, St. John's, St. Peter Port, Stanley, Stockholm, Sucre, Suva, Swansea, Swindon, Sydney, T'bilisi, Taipei, Tallahassee, Tallinn, Tarawa, Tashkent, Tbilisi, Tegucigalpa, Tehran, Thimphu, Tirana, Tirane, Tokyo, Toledo, Topeka, Torshavn, Trenton, Tripoli, Tunis, Ulaanbataar, Ulan Bator, Vaduz, Valletta, Vienna, Vientiane, Vilnius, Virginia Beach, W. Indies, Warsaw, Washington DC, Wellington, Wichita, Willemstad, Wilmington, Windhoek, Wolverhampton, Yamoussoukro, Yangon, Yaounde, Yaren, Yerevan, Zagreb

US Cities

Albany, Albuquerque, Anchorage, Annapolis, Atlanta, Augusta, Austin, Baltimore, Baton Rouge, Billings, Birmingham, Bismarck, Boise, Boston, Bridgeport, Buffalo, Burlington, Carson City, Charleston, Charlotte, Cheyenne, Chicago, Cleveland, Columbia, Columbus, Concord, Dallas, Denver, Des Moines, Detroit, Dover, Fargo, Frankfort, Harrisburg, Hartford, Helena, Honolulu, Houston, Indianapolis, Jackson, Jacksonville, Jefferson City, Juneau, Kansas City, Lansing, Las Vegas, Lincoln, Little Rock, Los Angeles, Louisville, Madison, Manchester, Memphis, Miami, Milwaukee, Minneapolis, Montgomery, Montpelier, Nashville, New Orleans, New York, Newark, Oklahoma City, Olympia, Omaha, Orlando, Philadelphia, Phoenix, Pierre, Portland, Portland, Providence, Raleigh, Richmond, Sacramento, Saint Paul, Salem, Salt Lake City, San Diego, San Francisco, Santa Fe, Seattle, Sioux Falls, Springfield, Tallahassee, Toledo, Topeka, Trenton, Virginia Beach, Wichita, Wilmington

CHAPTER 7

Thanks

The sun calculations in this module were adapted, for Python, from the spreadsheets on the following page.

> https://www.esrl.noaa.gov/gmd/grad/solcalc/calcdetails.html

The moon phase calculation is based on some javascript code from Sky and Telescope magazine

> Moon-phase calculation
> Roger W. Sinnott, Sky & Telescope, June 16, 2006.

Also to Sphinx for making doc generation an easy thing (not that the writing of the docs is any easier.)

CHAPTER 8

Contact

Simon Kennedy <sffjunkie+code@gmail.com>

CHAPTER 9

Version History

Version	Description
1.4.1	• Using versioneer to manage version numbers
1.4	• Changed to use calculations from NOAA spreadsheets • Changed some exception error messages for when sun does not reach a requested elevation. • Added more tests
1.3.4	• Changes to project configuration files. No user facing changes.
1.3.3	• Fixed call to twilight_utc as date and direction parameters were reversed.
1.3.2	• Updated URL to point to gitgub.com • Added Apache 2.0 boilerplate to source file
1.3.1	• Added LICENSE file to sdist
1.3	• Corrected solar zenith to return the angle from the vertical. • Added solar midnight calculation.
1.2	• Added handling for when unicode literals are used. This may possibly affect your code if you're using Python 2 (there are tests for this but they may not catch all uses.) (Bug 1588198) • Changed timezone for Phoenix, AZ to America/Phoenix (Bug 1561258)
1.1	• Added methods to calculate Twilight, the Golden Hour and the Blue Hour.

The `astral` Module

The `astral` module provides the means to calculate dawn, sunrise, solar noon, sunset, dusk and rahukaalam times, plus solar azimuth and elevation, for specific locations or at a specific latitude/longitude. It can also calculate the moon phase for a specific date.

The module provides 2 main classes `Astral` and `Location`.

`Astral` Has 2 main responsibilities

- Calculates the events in the UTC timezone.
- Provides access to location data

`Location` Holds information about a location and provides functions to calculate the event times for the location in the correct time zone.

For example

```
>>> from astral import *
>>> a = Astral()
>>> location = a['London']
>>> print('Information for %s' % location.name)
Information for London
>>> timezone = location.timezone
>>> print('Timezone: %s' % timezone)
Timezone: Europe/London
>>> print('Latitude: %.02f; Longitude: %.02f' % (location.latitude,
... location.longitude))
Latitude: 51.60; Longitude: 0.05
>>> from datetime import date
>>> d = date(2009,4,22)
>>> sun = location.sun(local=True, date=d)
>>> print('Dawn:    %s' % str(sun['dawn']))
Dawn:    2009-04-22 05:12:56+01:00
```

The module currently provides 2 methods of obtaining location information; `AstralGeocoder` (the default, which uses information from within the module) and `GoogleGeocoder` (which obtains information from Google's Map Service.)

To use the `GoogleGeocoder` pass the class as the *geocoder* parameter to `Astral.__init__()` or by setting the *geocoder* property to an instance of `GoogleGeocoder`:

```
>>> from astral import GoogleGeocoder
>>> a = Astral(GoogleGeocoder)
```

or

```
>>> from astral import GoogleGeocoder
>>> a = Astral()
>>> a.geocoder = GoogleGeocoder()
```

Astral

class `astral.`**`Astral`**(*geocoder=<class 'astral.AstralGeocoder'>*)

`__getitem__`(*key*)
Returns the Location instance specified by `key`.

`solar_depression`
The number of degrees the sun must be below the horizon for the dawn/dusk calculation.

Can either be set as a number of degrees below the horizon or as one of the following strings

String	Degrees
civil	6.0
nautical	12.0
astronomical	18.0

`sun_utc`(*date*, *latitude*, *longitude*)
Calculate all the info for the sun at once. All times are returned in the UTC timezone.

Parameters

- **`date`** (`datetime.date`) – Date to calculate for.
- **`latitude`** (*`float`*) – Latitude - Northern latitudes should be positive
- **`longitude`** (*`float`*) – Longitude - Eastern longitudes should be positive

Returns Dictionary with keys `dawn`, `sunrise`, `noon`, `sunset` and `dusk` whose values are the results of the corresponding *_utc* methods.

Return type dict

`dawn_utc`(*date*, *latitude*, *longitude*, *depression=0*)
Calculate dawn time in the UTC timezone.

Parameters

- **`date`** (`datetime.date`) – Date to calculate for.
- **`latitude`** (*`float`*) – Latitude - Northern latitudes should be positive
- **`longitude`** (*`float`*) – Longitude - Eastern longitudes should be positive
- **`depression`** (*`float`*) – Override the depression used

Returns The UTC date and time at which dawn occurs.

Return type `datetime`

`sunrise_utc`(*date*, *latitude*, *longitude*)
Calculate sunrise time in the UTC timezone.

Parameters

- **`date`** (`datetime.date`) – Date to calculate for.
- **`latitude`** (*`float`*) – Latitude - Northern latitudes should be positive
- **`longitude`** (*`float`*) – Longitude - Eastern longitudes should be positive

Returns The UTC date and time at which sunrise occurs.

Return type `datetime`

`solar_noon_utc`(*date*, *longitude*)
Calculate solar noon time in the UTC timezone.

Parameters

- **`date`** (`datetime.date`) – Date to calculate for.

- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns The UTC date and time at which noon occurs.

Return type datetime

sunset_utc(*date*, *latitude*, *longitude*)
Calculate sunset time in the UTC timezone.

Parameters

- **date** (datetime.date) – Date to calculate for.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns The UTC date and time at which sunset occurs.

Return type datetime

dusk_utc(*date*, *latitude*, *longitude*, *depression=0*)
Calculate dusk time in the UTC timezone.

Parameters

- **date** (datetime.date) – Date to calculate for.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive
- **depression** (*float*) – Override the depression used

Returns The UTC date and time at which dusk occurs.

Return type datetime

solar_midnight_utc(*date*, *longitude*)
Calculate solar midnight time in the UTC timezone.

Note that this claculates the solar midgnight that is closest to 00:00:00 of the specified date i.e. it may return a time that is on the previous day.

Parameters

- **date** (datetime.date) – Date to calculate for.
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns The UTC date and time at which midnight occurs.

Return type datetime

daylight_utc(*date*, *latitude*, *longitude*)
Calculate daylight start and end times in the UTC timezone.

Parameters

- **date** (datetime.date) – Date to calculate for.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns A tuple of the UTC date and time at which daylight starts and ends.

Return type (datetime, datetime)

`night_utc`(*date*, *latitude*, *longitude*)
Calculate night start and end times in the UTC timezone.

Night is calculated to be between astronomical dusk on the date specified and astronomical dawn of the next day.

Parameters

- **date** (`datetime.date`) – Date to calculate for.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns A tuple of the UTC date and time at which night starts and ends.

Return type (`datetime`, `datetime`)

`twilight_utc`(*direction*, *date*, *latitude*, *longitude*)
Returns the start and end times of Twilight in the UTC timezone when the sun is traversing in the specified direction.

This method defines twilight as being between the time when the sun is at -6 degrees and sunrise/sunset.

Parameters

- **direction** (*int*) – Determines whether the time is for the sun rising or setting. Use `astral.SUN_RISING` or `astral.SUN_SETTING`.
- **date** (`datetime.date`) – The date for which to calculate the times.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns A tuple of the UTC date and time at which twilight starts and ends.

Return type (`datetime`, `datetime`)

`golden_hour_utc`(*direction*, *date*, *latitude*, *longitude*)
Returns the start and end times of the Golden Hour in the UTC timezone when the sun is traversing in the specified direction.

This method uses the definition from PhotoPills i.e. the golden hour is when the sun is between 4 degrees below the horizon and 6 degrees above.

Parameters

- **direction** (*int*) – Determines whether the time is for the sun rising or setting. Use `astral.SUN_RISING` or `astral.SUN_SETTING`.
- **date** (`datetime.date`) – The date for which to calculate the times.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns A tuple of the UTC date and time at which the Golden Hour starts and ends.

Return type (`datetime`, `datetime`)

`blue_hour_utc`(*direction*, *date*, *latitude*, *longitude*)
Returns the start and end times of the Blue Hour in the UTC timezone when the sun is traversing in the specified direction.

This method uses the definition from PhotoPills i.e. the blue hour is when the sun is between 6 and 4 degrees below the horizon.

Parameters

- **`direction`** (*`int`*) – Determines whether the time is for the sun rising or setting. Use `astral.SUN_RISING` or `astral.SUN_SETTING`.
- **`date`** (`datetime.date`) – The date for which to calculate the times.
- **`latitude`** (*`float`*) – Latitude - Northern latitudes should be positive
- **`longitude`** (*`float`*) – Longitude - Eastern longitudes should be positive

Returns A tuple of the UTC date and time at which the Blue Hour starts and ends.

Return type (`datetime`, `datetime`)

`time_at_elevation_utc`(*elevation*, *direction*, *date*, *latitude*, *longitude*)
Calculate the time in the UTC timezone when the sun is at the specified elevation on the specified date.

Note: This method uses positive elevations for those above the horizon.

Parameters

- **`elevation`** (*`float`*) – Elevation in degrees above the horizon to calculate for.
- **`direction`** (*`int`*) – Determines whether the calculated time is for the sun rising or setting. Use `astral.SUN_RISING` or `astral.SUN_SETTING`. Default is rising.
- **`date`** (`datetime.date`) – Date to calculate for.
- **`latitude`** (*`float`*) – Latitude - Northern latitudes should be positive
- **`longitude`** (*`float`*) – Longitude - Eastern longitudes should be positive

Returns The UTC date and time at which the sun is at the required elevation.

Return type `datetime`

`solar_azimuth`(*dateandtime*, *latitude*, *longitude*)
Calculate the azimuth angle of the sun.

Parameters

- **`dateandtime`** (`datetime`) – The date and time for which to calculate the angle.
- **`latitude`** (*`float`*) – Latitude - Northern latitudes should be positive
- **`longitude`** (*`float`*) – Longitude - Eastern longitudes should be positive

Returns The azimuth angle in degrees clockwise from North.

Return type float

If *dateandtime* is a naive Python datetime then it is assumed to be in the UTC timezone.

`solar_elevation`(*dateandtime*, *latitude*, *longitude*)
Calculate the elevation angle of the sun.

Parameters

- **`dateandtime`** (`datetime`) – The date and time for which to calculate the angle.
- **`latitude`** (*`float`*) – Latitude - Northern latitudes should be positive
- **`longitude`** (*`float`*) – Longitude - Eastern longitudes should be positive

Returns The elevation angle in degrees above the horizon.

Return type float

If *dateandtime* is a naive Python datetime then it is assumed to be in the UTC timezone.

solar_zenith(*dateandtime*, *latitude*, *longitude*)

Calculates the solar zenith angle.

Parameters

- **dateandtime** (datetime) – The date and time for which to calculate the angle.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns The zenith angle in degrees from vertical.

Return type float

If *dateandtime* is a naive Python datetime then it is assumed to be in the UTC timezone.

moon_phase(*date*)

Calculates the phase of the moon on the specified date.

Parameters **date** (datetime.date) – The date to calculate the phase for.

Returns

A number designating the phase

0 = New moon
7 = First quarter
14 = Full moon
21 = Last quarter

Return type int

rahukaalam_utc(*date*, *latitude*, *longitude*)

Calculate ruhakaalam times in the UTC timezone.

Parameters

- **date** (datetime.date) – Date to calculate for.
- **latitude** (*float*) – Latitude - Northern latitudes should be positive
- **longitude** (*float*) – Longitude - Eastern longitudes should be positive

Returns Tuple containing the start and end times for Rahukaalam.

Return type tuple

Location

class astral.**Location**(*info=None*)

Provides access to information for single location.

__init__(*info=None*)

Initializes the object with a tuple of information.

Parameters **info** – A tuple of information to fill in the location info.

The tuple should contain items in the following order

Field	Default
name	Greenwich
region	England
latitude	51.168
longitude	0
time zone name	Europe/London
elevation	24

See `timezone` property for a method of obtaining time zone names

`latitude`
The location's latitude

`latitude` can be set either as a string or as a number

For strings they must be of the form

degrees°minutes'[N|S] e.g. 51°31'N

For numbers, positive numbers signify latitudes to the North.

`longitude`
The location's longitude.

`longitude` can be set either as a string or as a number

For strings they must be of the form

degrees°minutes'[E|W] e.g. 51°31'W

For numbers, positive numbers signify longitudes to the East.

`elevation`
The elevation in metres above sea level.

`timezone`
The name of the time zone for the location.

A list of time zone names can be obtained from pytz. For example.

```
>>> from pytz import all_timezones
>>> for timezone in all_timezones:
...     print(timezone)
```

`tz`
Time zone information.

`tzinfo`
Time zone information.

`solar_depression`
The number of degrees the sun must be below the horizon for the dawn/dusk calculation.

Can either be set as a number of degrees below the horizon or as one of the following strings

String	Degrees
civil	6.0
nautical	12.0
astronomical	18.0

`sun`(*date=None*, *local=True*)
Returns dawn, sunrise, noon, sunset and dusk as a dictionary.

Parameters

- **date** – The date for which to calculate the times. If no date is specified then the current date will be used.
- **local** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns Dictionary with keys `dawn`, `sunrise`, `noon`, `sunset` and `dusk` whose values are the results of the corresponding methods.

Return type dict

dawn (*date=None*, *local=True*)
Calculates the time in the morning when the sun is a certain number of degrees below the horizon. By default this is 6 degrees but can be changed by setting the *Astral.solar_depression* property.

Parameters

- **date** – The date for which to calculate the dawn time. If no date is specified then the current date will be used.
- **local** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns The date and time at which dawn occurs.

Return type `datetime`

sunrise (*date=None*, *local=True*)
Return sunrise time.

Calculates the time in the morning when the sun is a 0.833 degrees below the horizon. This is to account for refraction.

Parameters

- **date** – The date for which to calculate the sunrise time. If no date is specified then the current date will be used.
- **local** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns The date and time at which sunrise occurs.

Return type `datetime`

solar_noon (*date=None*, *local=True*)
Calculates the solar noon (the time when the sun is at its highest point.)

Parameters

- **date** – The date for which to calculate the noon time. If no date is specified then the current date will be used.
- **local** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns The date and time at which the solar noon occurs.

Return type `datetime`

sunset (*date=None*, *local=True*)
Calculates sunset time (the time in the evening when the sun is a 0.833 degrees below the horizon. This is to account for refraction.)

Parameters

- **`date`** – The date for which to calculate the sunset time. If no date is specified then the current date will be used.
- **`local`** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns The date and time at which sunset occurs.

Return type `datetime`

`dusk`(*date=None*, *local=True*)

Calculates the dusk time (the time in the evening when the sun is a certain number of degrees below the horizon. By default this is 6 degrees but can be changed by setting the *`solar_depression`* property.)

Parameters

- **`date`** – The date for which to calculate the dusk time. If no date is specified then the current date will be used.
- **`local`** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns The date and time at which dusk occurs.

Return type `datetime`

`daylight`(*date=None*, *local=True*)

Calculates the daylight time (the time between sunrise and sunset)

Parameters

- **`date`** – The date for which to calculate daylight. If no date is specified then the current date will be used.
- **`local`** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns A tuple containing the start and end times

Return type tuple(`datetime`, `datetime`)

`night`(*date=None*, *local=True*)

Calculates the night time (the time between astronomical dusk and astronomical dawn of the next day)

Parameters

- **`date`** – The date for which to calculate the start of the night time. If no date is specified then the current date will be used.
- **`local`** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns A tuple containing the start and end times

Return type tuple(`datetime`, `datetime`)

`twilight`(*direction=SUN_RISING*, *date=None*, *local=True*)

Returns the start and end times of Twilight in the UTC timezone when the sun is traversing in the specified direction.

This method defines twilight as being between the time when the sun is at -6 degrees and sunrise/sunset.

Parameters

- **`direction`** (*`int`*) – Determines whether the time is for the sun rising or setting. Use `astral.SUN_RISING` or `astral.SUN_SETTING`.

- **date** (datetime.date) – The date for which to calculate the times.
- **local** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns A tuple of the UTC date and time at which twilight starts and ends.

Return type (datetime, datetime)

golden_hour(*direction=SUN_RISING*, *date=None*, *local=True*)

Returns the start and end times of the Golden Hour when the sun is traversing in the specified direction.

This method uses the definition from PhotoPills i.e. the golden hour is when the sun is between 4 degrees below the horizon and 6 degrees above.

Parameters

- **direction** (*int*) – Determines whether the time is for the sun rising or setting. Use astral.SUN_RISING or astral.SUN_SETTING. Default is rising.
- **date** (datetime.date) – The date for which to calculate the times.
- **local** – True = Times to be returned in location's time zone; False = Times to be returned in UTC. If not specified then the time will be returned in local time

Returns A tuple of the date and time at which the Golden Hour starts and ends.

Return type (datetime, datetime)

blue_hour(*direction=SUN_RISING*, *date=None*, *local=True*)

Returns the start and end times of the Blue Hour when the sun is traversing in the specified direction.

This method uses the definition from PhotoPills i.e. the blue hour is when the sun is between 6 and 4 degrees below the horizon.

Parameters

- **direction** (*int*) – Determines whether the time is for the sun rising or setting. Use astral.SUN_RISING or astral.SUN_SETTING. Default is rising.
- **date** – The date for which to calculate the times. If no date is specified then the current date will be used.
- **local** – True = Times to be returned in location's time zone; False = Times to be returned in UTC. If not specified then the time will be returned in local time

Returns A tuple of the date and time at which the Blue Hour starts and ends.

Return type (datetime, datetime)

time_at_elevation(*elevation*, *direction=SUN_RISING*, *date=None*, *local=True*)

Calculate the time when the sun is at the specified elevation.

Note: This method uses positive elevations for those above the horizon.

Elevations greater than 90 degrees are converted to a setting sun i.e. an elevation of 110 will calculate a setting sun at 70 degrees.

Parameters

- **elevation** (*float*) – Elevation in degrees above the horizon to calculate for.
- **direction** (*int*) – Determines whether the time is for the sun rising or setting. Use astral.SUN_RISING or astral.SUN_SETTING. Default is rising.

- **`date`** – The date for which to calculate the elevation time. If no date is specified then the current date will be used.
- **`local`** – True = Time to be returned in location's time zone; False = Time to be returned in UTC. If not specified then the time will be returned in local time

Returns The date and time at which dusk occurs.

Return type `datetime`

`solar_azimuth`(*dateandtime=None*)

Calculates the solar azimuth angle for a specific date/time.

Parameters **`dateandtime`** (`datetime`) – The date and time for which to calculate the angle.

Returns The azimuth angle in degrees clockwise from North.

Return type float

`solar_elevation`(*dateandtime=None*)

Calculates the solar elevation angle for a specific time.

Parameters **`dateandtime`** (`datetime`) – The date and time for which to calculate the angle.

Returns The elevation angle in degrees above the horizon.

Return type float

`solar_zenith`(*dateandtime=None*)

Calculates the solar zenith angle for a specific time.

Parameters **`dateandtime`** (`datetime`) – The date and time for which to calculate the angle.

Returns The zenith angle in degrees from vertical.

Return type float

`moon_phase`(*date=None*)

Calculates the moon phase for a specific date.

Parameters **`date`** (`datetime.date`) – The date to calculate the phase for. If ommitted the current date is used.

Returns

A number designating the phase

0 = New moon

7 = First quarter

14 = Full moon

21 = Last quarter

Return type int

`rahukaalam`(*date=None*, *local=True*)

Calculates the period of rahukaalam.

Parameters

- **`date`** – The date for which to calculate the rahukaalam period. A value of `None` uses the current date.
- **`local`** – True = Time to be returned in location's time zone; False = Time to be returned in UTC.

Returns Tuple containing the start and end times for Rahukaalam.

Return type tuple

Geocoders

class `astral.`**`AstralGeocoder`**

Looks up geographic information from the locations stored within the module

class `astral.`**`GoogleGeocoder`**`(`*cache=False*`)`

Use Google Maps API Web Service to lookup GPS co-ordinates, timezone and elevation.

See the following for more info. https://developers.google.com/maps/documentation/

Python Module Index

Index