Julian date python module

The `juliandate` module is designed to simplify the complexities in performing mathematical operations on different dates, while seamlessly interfacing with python's own datetime module.

The `juliandate` module can be fully functional on its own if you wish to only be dependent on a single module. It can also interface with some of the objects found in python's standard `datetime` module if your project uses them, or you wish to take advantage of their functionality.

Aware and Naive

According to the python documentation:

> Date and time objects may be categorized as "aware" or "naive" depending on whether or not they include timezone information.

Using these definitions the `JulianDate` object can also be either aware or naive. If an object is constructed with the timezone set, the object will be aware, otherwise the object is naive.

The one ambiguity here is when a `JulianDate` object is purposely constructed to be in the GMT timezone. While the object is aware of its timezone, internally there is no difference between a timezone of +0 and a naive object. However, the user can be certain that any computations done with the former object are accurate and valid.

Note on time zones:

Unless stated otherwise, a time zone parameter refers to either an integer or floating point number representing the current time zone offset from UTC (including DST).

Constants

There is a single constant from the `juliandate` module which represents the J2000 epoch:

`juliandate.` **J2000**:

- The J2000 epoch equivalent to `JulianDate(1, 1, 2000, 12, 0, 0, timeZone=0)`

Types

class `juliandate.` **JulianDate**:

- The JulianDate class represents an instance in time and is composed of a day number and fraction of the next day.

Methods

There is a single module level method which allows the creation of a `JulianDate` object with the current time.

method `juliandate.` **now(timezone=None)**:

- Return a new `JulianDate` object from the current time using python's standard modules, therefore the implementation may be dependent on your python version.
- If the timezone parameter is `None` the timezone offset is also computed using python's standard modules, otherwise the current time is computed and adjusted to accommodate timezone parameter.

JulianDate Objects

A `JulianDate` object represents a single instance in time by converting a calendar date into a Julian day number. A Julian date consists of the day number, or the whole number of days after a predetermined date, and the fraction of the current day after 12:00:00 UT1. The day number and day fraction are stored in two different variables to maintain precision of the day fraction, as the modern day numbers exceed 2460000. Therefore, the precision of the fraction part of the day can be guaranteed to be the same precision as a floating point number in your python version.

The bulk of the `JulianDate` class handles converting from a date/time to a Julian date and back to its date and time components, as well as handling the mathematical operations involved between instances and other python classes.

Constructor:

class `juliandate.` **JulianDate(month, day, year, hour, minute, second, timezone=0)**

- All arguments except timezone are required. timezone can be an integer or floating point number representing the offset of the timezone relative to UTC. seconds can be an integer or floating point number. All other parameters must be integers and in the following ranges:
  - 1 <= month <= 12
  - 1 <= day <= number of days in the given month and year
  - -inf < year < +inf
  - 0 <= hour <= 59
  - 0 <= minute <= 59
  - 0.0 <= second < 60.0

- A ValueError is raised if any argument is outside these values.

Other Constructors:

classmethod `JulianDate.` **fromNumber(number, timezone=0)**

- Creates a `JulianDate` directly from a known Julian day number.
- timezone can be an integer or floating point number representing the offset from UTC.
- If the parameters are not integer or floating point numbers then TypeError is raised.

classmethod `juliandate.` **fromDatetime(date)**

- Creates a `JulianDate` from a `datetime` instance from python's `datetime` module.
- If the date parameter is not a `datetime` type then TypeError is raised.

Instance attributes (read-only):

`JulianDate.` **number**

- The integer part of the Julian date, the Julian day number

`JulianDate.` **fraction**

- The fraction part of the Julian date

`JulianDate.` **value**

- The Julian date, equivalent to `JulianDate.number + JulianDate.fraction`

`JulianDate.` **timezone**

- If the object is naive 0, otherwise the UTC offset

Supported Operations:

Operation	Parameters	Result
addition (1)	`timedelta`	`JulianDate`
subtraction (2)	`JulianDate`	`float`
==, !=, <, <=, >, >=	`JulianDate`	`bool`

1. The __add__ and __radd__ methods support addition of a `timedelta` object from python's `datetime` module and returns a new `JulianDate` object. To add an integer or floating point number to a `JulianDate` object, see `JulianDate.future()` .
2. The __sub__ method supports subtraction between `JulianDate` objects and returns a floating point number. To find the difference between a `JulianDate` and a number representation of a Julian date see `JulianDate.difference()` .

Instance Methods:

`JulianDate.` **setTime(month, day, year, hour, minute, second, timezone=0)**

- Updates the current value of the Julian date. Useful for changing a Julian day value without needing to create a new instance.
- All arguments except timezone are required. timezone can be an integer or floating point number representing the offset of the timezone relative to UTC. seconds can be an integer or floating point number. All other parameters must be integers and in the following ranges:
  - 1 <= month <= 12
  - 1 <= day <= number of days in the given month and year
  - -inf < year < +inf
  - 0 <= hour <= 59
  - 0 <= minute <= 59
  - 0.0 <= second < 60.0
- A ValueError is raised if any argument is outside these values.

`JulianDate.` **__iter__()**

- Yields a dictionary of the internal Julian day values with the keys `dayNumber` and `dayFraction` .

`JulianDate.` **__str__()**

- Creates a string of the date and time components in the format 'NUMBER.FRACTION --- MM/DD/YYYY HH:MM:SS.SSS +/-T.T UTC' where:
  - NUMBER is the Julian day number
  - FRACTION is the Julian date fraction
  - +/-T.T is a floating point number representing the UTC offset (e.g. -6.0 for CST)

`JulianDate.` **__repr__()**

- Returns a JSON string representation of the Julian day number parts, represented as `dayNumber` and `dayFraction` .

`JulianDate.` **future(days)**

- The numeric version of the addition operator. Computes a future (positive) or past (negative) `JulianDate` where days is in solar days, where a solar day is equal to 86400 seconds. Useful for addition without creating a `timedelta` object, equivalent to `JulianDate + timedelta(days=days)` .

`JulianDate.` **difference(jd)**

- The numeric version of the subtraction operator. Computes the difference between two Julian dates when the subtractend is an int or float, equivalent to `JulianDate - JulianDate.fromNumber(jd)` .

`JulianDate.` **date(timezone=None)**

- Returns a string of the date and time. If the `JulianDate` is aware, the timezone parameter is used to get the date/time in a different timezone, or the current timezone if set to None (default behaviour).

- > ⚠ If the JulianDate is not aware and timezone is not None the behavior may be unexpected and should not be trusted.

`JulianDate.` **day(timezone=None)**

- Returns a string of the day of the year. If the `JulianDate` is aware, the timezone parameter is used to get the date in a different timezone, or the current timezone if set to None (default behaviour).

- > ⚠ If the JulianDate is not aware and timezone is not None the behavior may be unexpected and should not be trusted.

`JulianDate.` **time(timezone=None)**

- Returns a string of the time. If the `JulianDate` is aware, the timezone parameter is used to get the time in a different timezone, or the current timezone if set to None (default behaviour).

- > ⚠ If the JulianDate is not aware and timezone is not None the behavior may be unexpected and should not be trusted.

Conversion methods:

`JulianDate.` **toDatetime()**

- Returns a `datetime` object with the equivalent date and time.