Delorean Documentation

Release 1.0.0

Mahdi Yusuf

Nov 21, 2018

Contents

datetime? Where we're going, we don't need datetime.

This document describes Delorean v1.0.0.

Delorean is the name of the car in the movie Back to the Future. The movie deals with a lot of time travel, hence the name Delorean as a module dealing with datetimes.

Delorean is a library for clearing up the inconvenient truths that arise dealing with datetimes in Python. Understanding that timing is a delicate enough of a problem *delorean* hopes to provide a cleaner less troublesome solution to shifting, manipulating, generating *datetimes*.

Delorean stands on the shoulders of giants pytz and dateutil

Delorean will provide natural language improvements for manipulating time, as well as datetime abstractions for ease of use. The overall goal is to improve datetime manipulations, with a little bit of software and philosophy.

Pretty much make you a badass, time traveller.

CHAPTER 1

Interface Update

Version 1.0.0 introduces the following breaking changes:

- *Delorean.epoch* is a property, not a function.
- *Delorean.midnight* is a property, not a function.
- *Delorean.naive* is a property, not a function.
- *Delorean.timezone* is a property, not a function.

Please make sure to update your code accordingly.

CHAPTER 2

Getting Started

Here is the world without a flux capacitor at your side.:

```
from datetime import datetime
import pytz

est = pytz.timezone('US/Eastern')
d = datetime.now(pytz.utc)
d = est.normalize(d.astimezone(est))
return d
```

Now lets warm up the *delorean*:

```
from delorean import Delorean

d = Delorean()
d = d.shift('US/Eastern')
return d
```

Look at you looking all fly. This was just a test drive checkout out what else *delorean* can help with below.

CHAPTER 3

Guide

3.1 License

3.2 Installation

This software can be installed in multiple ways. The ways that are recommended are as follows.

3.2.1 Obtaining the Source

This is a library targeted specifcally to developers so you will most likely want to get your hands on the source. It can be obtained as follows.:

```
$ git clone git://github.com/myusuf3/delorean.git
```

3.2.2 Installing from PyPi

I would personally recommend using pip but any of the following ways should work.

To install simply run:

```
$ pip install delorean
```

or if you must easy_install

```
$ easy_install delorean
```

3.2.3 Installing from Source

If you have a copy of the source simply running the following from the root directory should have the same effect.:

```
$ python setup.py install
```

3.3 Usage

Delorean aims to provide you with convient ways to get significant dates and times and easy ways to move dates from state to state.

In order to get the most of the documentation we will define some terminology.

1. **naive datetime** – a datetime object without a timezone.
2. **localized datetime** – a datetime object with a timezone.
3. **localizing** – associating a naive datetime object with a timezone.
4. **normalizing** – shifting a localized datetime object from one timezone to another, this changes both tzinfo and datetime object.

3.3.1 Making Some Time

Making time with *delorean* is much easier than in life.

Start with importing delorean:

```
>>> from delorean import Delorean
```

Now lets create a create *datetime* with the current datetime and UTC timezone

```
>>> d = Delorean()
>>> d
Delorean(datetime=datetime.datetime(2013, 1, 12, 6, 10, 33, 110674),  timezone='UTC')
```

Do you want to normalize this timezone to another timezone? Simply do the following

```
>>> d = d.shift("US/Eastern")
>>> d
Delorean(datetime=datetime.datetime(2013, 1, 12, 1, 10, 38, 102223), timezone='US/
↪Eastern')
```

Now that you have successfully shifted the timezone you can easily return a localized datetime object or date with ease.

```
>>> d.datetime
datetime.datetime(2013, 1, 12, 01, 10, 38, 102223, tzinfo=<DstTzInfo 'US/Eastern' EST-
↪1 day, 19:00:00 STD>)
>>> d.date
datetime.date(2013, 1, 12)
```

For the purists out there you can do things like so.

```
>>> d.naive
datetime.datetime(2013, 1, 12, 1, 10, 38, 102223)
>>> d.epoch
1357971038.102223
```

You can also create Delorean object using unix timestamps.

```
from delorean import epoch
>>> epoch(1357971038.102223).shift("US/Eastern")
Delorean(datetime=datetime.datetime(2013, 1, 12, 1, 10, 38, 102223), timezone='US/
↪Eastern')
```

As you can see *delorean* returns a Delorean object which you can shift to the appropriate timezone to get back your original datetime object from above.

Note: If you are comparing Delorean objects the time since epoch will be used internally for comparison. This allows for the greatest accuracy when comparing Delorean objects from different timezones!

Delorean also now accepts localized datetimes. This means if you had a previously localized datetime object, Delorean will now accept these values and set the associated timezone and datetime information on the Delorean object.

Note: If you pass in a timezone with a localized datetime the timezone will be ignored, since the datetime object you are passing already has timezone information already associated with it.

```
>>> tz = timezone("US/Pacific")
>>> dt = tz.localize(datetime.utcnow())
datetime.datetime(2013, 3, 16, 5, 28, 11, 536818, tzinfo=<DstTzInfo 'US/Pacific' PDT-
↪1 day, 17:00:00 DST>)
>>> d = Delorean(datetime=dt)
>>> d
Delorean(datetime=datetime.datetime(2013, 3, 16, 5, 28, 11, 536818), timezone='US/
↪Pacific')
>>> d = Delorean(datetime=dt, timezone="US/Eastern")
>>> d
Delorean(datetime=datetime.datetime(2013, 3, 16, 5, 28, 11, 536818), timezone='US/
↪Pacific')
```

3.3.2 Time Arithmetic

Delorean can also handle timedelta arithmetic. A timedelta may be added to or subtracted from a *Delorean* object. Additionally, you may subtract a *Delorean* object from another Delorean object to obtain the timedelta between them.

```
>>> d = Delorean()
>>> d
Delorean(datetime=datetime.datetime(2014, 6, 3, 19, 22, 59, 289779), timezone='UTC')
>>> d += timedelta(hours=2)
>>> d
Delorean(datetime=datetime.datetime(2014, 6, 3, 21, 22, 59, 289779), timezone='UTC')
>>> d - timedelta(hours=2)
Delorean(datetime=datetime.datetime(2014, 6, 3, 19, 22, 59, 289779), timezone='UTC')
>>> d2 = d + timedelta(hours=2)
>>> d2 - d
datetime.timedelta(0, 7200)
```

Delorean objects are considered equal if they represent the same time in UTC.

```
>>> d1 = Delorean(datetime(2015, 1, 1), timezone='US/Pacific')
>>> d2 = Delorean(datetime(2015, 1, 1, 8), timezone='UTC')
>>> d1 == d2
True
```

3.3.3 Natural Language

Delorean provides many ways to get certain date relative to another, often getting something simple like the next year or the next thursday can be quite troublesome.

Delorean provides several conveniences for this type of behaviour. For example if you wanted to get next Tuesday from today you would simply do the following

```
>>> d = Delorean()
>>> d
Delorean(datetime=datetime.datetime(2013, 1, 20, 19, 41, 6, 207481), timezone='UTC')
>>> d.next_tuesday()
Delorean(datetime=datetime.datetime(2013, 1, 22, 19, 41, 6, 207481), timezone='UTC')
```

Last Tuesday? Two Tuesdays ago at midnight? No problem.

```
>>> d.last_tuesday()
Delorean(datetime=datetime.datetime(2013, 1, 15, 19, 41, 6, 207481), timezone='UTC')
>>> d.last_tuesday(2).midnight
datetime.datetime(2013, 1, 8, 0, 0, tzinfo=<UTC>)
```

3.3.4 Replace Parts

Using the *replace* method on *Delorean* objects, we can replace the *hour*, *minute*, *second*, *year* etc like the the *replace* method on *datetime*.

```
>>> d = Delorean(datetime(2015, 1, 1, 12, 15), timezone='UTC')
>>> d.replace(hour=8)
Delorean(datetime=datetime.datetime(2015, 1, 1, 8, 15), timezone='UTC')
```

3.3.5 Truncation

Often we dont care how many milliseconds or even seconds that are present in our datetime object. For example it is a nuisance to retrieve *datetimes* that occur in the same minute. You would have to go through the annoying process of

replacing zero for the units you don't care for before doing a comparison.

Delorean comes with a method that allows you to easily truncate to different unit of time: millisecond, second, minute, hour, etc.

```
>>> d = Delorean()
>>> d
Delorean(datetime=datetime.datetime(2013, 1, 21, 3, 34, 30, 418069), timezone='UTC')
>>> d.truncate('second')
Delorean(datetime=datetime.datetime(2013, 1, 21, 3, 34, 30), timezone='UTC')
>>> d.truncate('hour')
Delorean(datetime=datetime.datetime(2013, 1, 21, 3, 0), timezone='UTC')
```

Though it might seem obvious *delorean* also provides truncation to the month and year levels as well.

```
>>> d = Delorean(datetime=datetime(2012, 5, 15, 03, 50, 00, 555555), timezone="US/
↪Eastern")
>>> d
Delorean(datetime=datetime.datetime(2012, 5, 15, 3, 50, 0, 555555), timezone='US/
↪Eastern')
>>> d.truncate('month')
Delorean(datetime=datetime.datetime(2012, 5, 1), timezone='US/Eastern')
>>> d.truncate('year')
Delorean(datetime=datetime.datetime(2012, 1, 1), timezone='US/Eastern')
```

3.3.6 Strings and Parsing

Another pain is dealing with strings of datetimes. *Delorean* can help you parse all the datetime strings you get from various APIs.

```
>>> from delorean import parse
>>> parse("2011/01/01 00:00:00 -0700")
Delorean(datetime=datetime.datetime(2011, 1, 1, 7), timezone='UTC')
```

As shown above if the string passed has offset data *delorean* will convert the resulting object to UTC, if there is no timezone information passed in UTC is assumed.

Ambiguous cases

There might be cases where the string passed to parse is a bit ambiguous for example. In the case where *2013-05-06* is passed is this May 6th, 2013 or is June 5th, 2013?

Delorean makes the assumptions that `dayfirst=True` and `yearfirst=True` this will lead to the following precedence.

If dayfirst is True and yearfirst is True:

- YY-MM-DD
- DD-MM-YY
- MM-DD-YY

So for example with default parameters *Delorean* will return '2013-05-06' as May 6th, 2013.

```
>>> parse("2013-05-06")
Delorean(datetime=datetime.datetime(2013, 5, 6), timezone='UTC')
```

Here are the precedence for the remaining combinations of `dayfirst` and `yearfirst`.

If dayfirst is False and yearfirst is False:

- MM-DD-YY
- DD-MM-YY
- YY-MM-DD

If dayfirst is True and yearfirst is False:

- DD-MM-YY
- MM-DD-YY
- YY-MM-DD

If dayfirst is False and yearfirst is True:

- YY-MM-DD
- MM-DD-YY
- DD-MM-YY

3.3.7 Making A Few Stops

Delorean wouldn't be complete without making a few stop in all the right places.

```
>>> import delorean
>>> from delorean import stops
>>> for stop in stops(freq=delorean.HOURLY, count=10):    print stop
...
Delorean(datetime=datetime.datetime(2013, 1, 21, 6, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 7, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 8, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 9, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 10, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 11, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 12, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 13, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 14, 25, 33), timezone='UTC')
Delorean(datetime=datetime.datetime(2013, 1, 21, 15, 25, 33), timezone='UTC')
```

This allows you to do clever composition like daily, hourly, etc. This method is a generator that produces *Delorean* objects. Excellent for things like getting every Tuesday for the next 10 weeks, or every other hour for the next three months.

With Power Comes

Now that you can do this you can also specify `timezones` as well `start` and `stop` dates for iteration.

```
>>> import delorean
>>> from delorean import stops
>>> from datetime import datetime
>>> d1 = datetime(2012, 5, 06)
>>> d2 = datetime(2013, 5, 06)
```

Note: The `stops` method only accepts naive datetime `start` and `stop` values.

Now in the case where you provide *timezone*, *start*, and *stop* all is good in the world!

```
>>> for stop in stops(freq=delorean.DAILY, count=10, timezone="US/Eastern", start=d1,␣
↪stop=d2):    print stop
...
Delorean(datetime=datetime.datetime(2012, 5, 6), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 7), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 8), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 9), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 10), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 11), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 12), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 13), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 14), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2012, 5, 15), timezone='US/Eastern')
```

Note: if no `start` or `timezone` value is specified start is assumed to be localized UTC object. If timezone is provided a normalized UTC to the correct timezone.

Now in the case where a naive stop value is provided you can see why the follow error occurs if you take into account the above note.

```
>>> for stop in stops(freq=delorean.DAILY, timezone="US/Eastern", stop=d2):    print␣
↪stop
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "delorean/interface.py", line 63, in stops
    bysecond=None, until=until, dtstart=start):
TypeError: can't compare offset-naive and offset-aware datetimes
```

You will be better off in scenarios of this nature to skip using either and use count to limit the range of the values returned.

```
>>> from delorean import stops
>>> for stop in stops(freq=delorean.DAILY, count=2, timezone="US/Eastern"):    print␣
↪stop
...
Delorean(datetime=datetime.datetime(2013, 1, 22, 0, 10, 10), timezone='US/Eastern')
Delorean(datetime=datetime.datetime(2013, 1, 23, 0, 10, 10), timezone='US/Eastern')
```

3.4 Interface

3.4.1 Delorean

class delorean.**Delorean**(*datetime=None*, *timezone=None*)

The class *Delorean <Delorean>* object. This method accepts naive datetime objects, with a string timezone.

date

Returns the actual date object associated with the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1, 12, 15), timezone='US/Pacific')
>>> d.date
datetime.date(2015, 1, 1)
```

datetime
Returns the actual datetime object associated with the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1, 12, 15), timezone='UTC')
>>> d.datetime
datetime.datetime(2015, 1, 1, 12, 15, tzinfo=<UTC>)
```

end_of_day
Returns the end of the day for the datetime assocaited with the Delorean object, modifying the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1, 12), timezone='UTC')
>>> d.end_of_day
datetime.datetime(2015, 1, 1, 23, 59, 59, 999999, tzinfo=<UTC>)
```

epoch
Returns the total seconds since epoch associated with the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1), timezone='US/Pacific')
>>> d.epoch
1420099200.0
```

format_datetime(*format='medium'*, *locale='en_US'*)
Return a date string formatted to the given pattern.

```
>>> d = Delorean(datetime(2015, 1, 1, 12, 30), timezone='US/Pacific')
>>> d.format_datetime(locale='en_US')
u'Jan 1, 2015, 12:30:00 PM'

>>> d.format_datetime(format='long', locale='de_DE')
u'1. Januar 2015 12:30:00 -0800'
```

Parameters

- **format** – one of "full", "long", "medium", "short", or a custom datetime pattern
- **locale** – a locale identifier

humanize()
Humanize relative to now:

```
>>> past = Delorean.utcnow() - timedelta(hours=1)
>>> past.humanize()
'an hour ago'
```

midnight
Returns midnight for datetime associated with the Delorean object modifying the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1, 12), timezone='UTC')
>>> d.midnight
datetime.datetime(2015, 1, 1, 0, 0, tzinfo=<UTC>)
```

naive
Returns a naive datetime object associated with the Delorean object, this method simply converts the localize datetime to UTC and removes the tzinfo that is associated with it modifying the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1), timezone='US/Pacific')
>>> d.naive
datetime.datetime(2015, 1, 1, 8, 0)
```

replace(***kwargs*)
Returns a new Delorean object after applying replace on the existing datetime object.

```
>>> d = Delorean(datetime(2015, 1, 1, 12, 15), timezone='UTC')
>>> d.replace(hour=8)
Delorean(datetime=datetime.datetime(2015, 1, 1, 8, 15), timezone='UTC')
```

shift(*timezone*)
Shifts the timezone from the current timezone to the specified timezone associated with the Delorean object, modifying the Delorean object and returning the modified object.

```
>>> d = Delorean(datetime(2015, 1, 1), timezone='US/Pacific')
>>> d.shift('UTC')
Delorean(datetime=datetime.datetime(2015, 1, 1, 8, 0), timezone='UTC')
```

start_of_day
Returns the start of the day for datetime assoicated with the Delorean object, modifying the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1, 12), timezone='UTC')
>>> d.start_of_day
datetime.datetime(2015, 1, 1, 0, 0, tzinfo=<UTC>)
```

timezone
Returns a valid tzinfo object associated with the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1), timezone='UTC')
>>> d.timezone
<UTC>
```

truncate(*s*)
Truncate the delorian object to the nearest s (second, minute, hour, day, month, year)

This is a destructive method, modifies the internal datetime object associated with the Delorean object.

```
>>> d = Delorean(datetime(2015, 1, 1, 12, 10), timezone='US/Pacific')
>>> d.truncate('hour')
Delorean(datetime=datetime.datetime(2015, 1, 1, 12, 0), timezone='US/Pacific')
```

exception delorean.**DeloreanInvalidDatetime**(*msg*)
Exception that is raised when an improper datetime object is passed in.

exception delorean.**DeloreanInvalidTimezone**(*msg*)
Exception that is raised when an invalid timezone is passed in.

delorean.**datetime_timezone**(*tz*)
This method given a timezone returns a localized datetime object.

delorean.**localize**(*dt*, *tz*)
Given a naive datetime object this method will return a localized datetime object

delorean.**move_datetime_month**(*dt*, *direction*, *num_shifts*)
Move datetime 1 month in the chosen direction. unit is a no-op, to keep the API the same as the day case

delorean.**move_datetime_week**(*dt*, *direction*, *num_shifts*)
Move datetime 1 week in the chosen direction. unit is a no-op, to keep the API the same as the day case

delorean.**move_datetime_year**(*dt*, *direction*, *num_shifts*)
Move datetime 1 year in the chosen direction. unit is a no-op, to keep the API the same as the day case

delorean.**normalize**(*dt*, *tz*)
Given a object with a timezone return a datetime object normalized to the proper timezone.

This means take the give localized datetime and returns the datetime normalized to match the specificed timezone.

delorean.**now**(*timezone=None*)
Return a Delorean object for the current local date and time, setting the timezone to the local timezone of the caller by default.

Parameters **timezone** (*Optional[datetime.tzinfo]*) – A custom timezone to use when computing the time.

Return type delorean.dates.Delorean

delorean.**parse**(*datetime_str*, *timezone=None*, *isofirst=True*, *dayfirst=True*, *yearfirst=True*)
Parses a datetime string and returns a *Delorean* object.

Parameters

- **datetime_str** – The string to be interpreted into a *Delorean* object.
- **timezone** – Pass this parameter and the returned Delorean object will be normalized to this timezone. Any offsets passed as part of datetime_str will be ignored.
- **isofirst** – try to parse string as date in ISO format before everything else.
- **dayfirst** – Whether to interpret the first value in an ambiguous 3-integer date (ex. 01/05/09) as the day (True) or month (False). If yearfirst is set to True, this distinguishes between YDM and YMD.
- **yearfirst** – Whether to interpret the first value in an ambiguous 3-integer date (ex. 01/05/09) as the year. If True, the first number is taken to be the year, otherwise the last number is taken to be the year.

```
>>> parse('2015-01-01 00:01:02')
Delorean(datetime=datetime.datetime(2015, 1, 1, 0, 1, 2), timezone='UTC')
```

If a fixed offset is provided in the datetime_str, it will be parsed and the returned *Delorean* object will store a *pytz.FixedOffest* as it's timezone.

```
>>> parse('2015-01-01 00:01:02 -0800')
Delorean(datetime=datetime.datetime(2015, 1, 1, 0, 1, 2), timezone=pytz.
↪FixedOffset(-480))
```

If the timezone argument is supplied, the returned Delorean object will be in the timezone supplied. Any offsets in the datetime_str will be ignored.

```
>>> parse('2015-01-01 00:01:02 -0500', timezone='US/Pacific')
Delorean(datetime=datetime.datetime(2015, 1, 1, 0, 1, 2), timezone='US/Pacific')
```

If an unambiguous timezone is detected in the datetime string, a Delorean object with that datetime and timezone will be returned.

```
>>> parse('2015-01-01 00:01:02 PST')
Delorean(datetime=datetime.datetime(2015, 1, 1, 0, 1, 2), timezone='America/Los_
↪Angeles')
```

However if the provided timezone is ambiguous, parse will ignore the timezone and return a *Delorean* object in UTC time.

```
>>> parse('2015-01-01 00:01:02 EST')
Delorean(datetime=datetime.datetime(2015, 1, 1, 0, 1, 2), timezone='UTC')
```

delorean.**range_daily**(*start=None*, *stop=None*, *timezone='UTC'*, *count=None*)
This an alternative way to generating sets of Delorean objects with DAILY stops

delorean.**range_hourly**(*start=None*, *stop=None*, *timezone='UTC'*, *count=None*)
This an alternative way to generating sets of Delorean objects with HOURLY stops

delorean.**range_monthly**(*start=None*, *stop=None*, *timezone='UTC'*, *count=None*)
This an alternative way to generating sets of Delorean objects with MONTHLY stops

delorean.**range_yearly**(*start=None*, *stop=None*, *timezone='UTC'*, *count=None*)
This an alternative way to generating sets of Delorean objects with YEARLY stops

delorean.**stops**(*freq*, *interval=1*, *count=None*, *wkst=None*, *bysetpos=None*, *bymonth=None*, *bymonthday=None*, *byyearday=None*, *byeaster=None*, *byweekno=None*, *byweekday=None*, *byhour=None*, *byminute=None*, *bysecond=None*, *timezone='UTC'*, *start=None*, *stop=None*)
This will create a list of delorean objects the apply to setting possed in.

delorean.**utcnow**()
Return a Delorean object for the current UTC date and time, setting the timezone to UTC.

3.5 Contribution

Delorean is currently in active development and welcomes code improvements and bug fixes. For those of your interested in providing documentation to the lesser know parts would be equally awesome!

Pull down the source:

```
$ git clone git@github.com:myusuf3/delorean.git
```

Note: All code contributions should have adequate tests. *Delorean* has high test coverage we would like to keep this way. We provide an easy way to use *make* command that will run all tests with *nose* and *coverage*. Simply run the following command from the root directory:

```
$ make test
```

3.5.1 Style

If you decide to contribute code please follow naming conventions and style guides the code currently complies too.

When in doubt, be safe, follow PEP-8.

3.5.2 Documentation

Contributing to documentation is as simple as editing the specified file in the *docs* directory of the source. We welcome all code improvements.

We use Sphinx for building our documentation. That can be obtained here .

Note: We provide easy to use *make* command that will build the docs and open a copy locally in your browser. Simply run the following command from the root directory:

```
$ make doc
```

3.5.3 Reporting Issues

This is most important of all, if there are issues please let us know. So we can improve *delorean*. If you don't report it, we probably wont fix it.

All issues should be reported on Github here.

Python Module Index

d

Index