fleming Documentation

Release 0.4.6

Wes Kendall

Mar 28, 2018

Contents

1 Welcome to fleming's documentation! 1

2 Function Overview 3

3 Installation 5

4 Fleming 7
- 4.1 convert_to_tz . . . 7
- 4.2 add_timedelta . . . 8
- 4.3 floor . . . 9
- 4.4 ceil . . . 11
- 4.5 intervals . . . 13
- 4.6 unix_time . . . 15

5 Contributing 17
- 5.1 Running the tests . . . 17
- 5.2 Code Quality . . . 17
- 5.3 Code Styling . . . 17
- 5.4 Building the docs . . . 18
- 5.5 Release Checklist . . . 18
- 5.6 Vulnerability Reporting . . . 18

6 Release Notes 19
- 6.1 v0.4.6 . . . 19
- 6.2 v0.4.5 . . . 19
- 6.3 v0.4.4 . . . 19
- 6.4 v0.4.1 . . . 19

7 License 21

CHAPTER 1

Welcome to fleming's documentation!

Fleming contains a set of routines for doing datetime manipulation. Named after Sandford Fleming, the father of worldwide standard timezones, this package is meant to aid datetime manipulations with regards to timezones.

Fleming addresses some of the common difficulties with timezones and datetime objects, such as performing arithmetic and datetime truncation across a Daylight Savings Time border. It also provides utilities for generating date ranges and getting unix times with respect to timezones.

Fleming accepts pytz timezone objects as parameters, and it is assumed that the user has a basic understanding of pytz. Click here for more information about pytz.

CHAPTER 2

Function Overview

A brief description of each function in this package is below. More detailed descriptions and advanced usage of the functions follow after that. Click on the function names to go to their detailed descriptions.

- *convert_to_tz* : Converts a datetime object into a provided timezone.
- *add_timedelta* : Adds a timedelta to a datetime object.
- *floor* : Rounds a datetime object down to the previous time interval.
- *ceil* : Rounds a datetime object up to the next time interval.
- *intervals* : Gets a range of times at a given timedelta interval.
- *unix_time* : Returns a unix time stamp of a datetime object.

CHAPTER 3

Installation

To install the latest release, type:

```
pip install fleming
```

To install the latest code directly from source, type:

```
pip install git+git://github.com/ambitioninc/fleming.git
```

CHAPTER 4

Fleming

4.1 convert_to_tz

fleming.**convert_to_tz**(*dt*, *tz*, *return_naive=False*)

Converts a time into another timezone.

Given an aware or naive datetime dt, convert it to the timezone tz.

Parameters

- **dt** (*datetime*) – A naive or aware datetime object. If dt is naive, it has UTC set as its timezone.
- **tz** (*pytz timezone*) – A pytz timezone object specifying the timezone to which dt should be converted.
- **return_naive** (*bool*) – A boolean describing whether the return value should be a naive datetime object.

Return type datetime

Returns An aware datetime object that was the result of converting dt into tz. If return_naive is True, the returned value has no tzinfo set.

```
>>> import fleming
>>> import datetime
>>> import pytz

>>> dt = datetime.datetime(2013, 2, 4)
>>> print dt
2013-02-04 00:00:00

>>> # Convert naive UTC time to aware EST time
>>> dt = fleming.convert_to_tz(dt, pytz.timezone('US/Eastern'))
>>> print dt
2013-02-03 19:00:00-05:00
```

```
>>> # Convert aware EST time to aware CST time
>>> dt = fleming.convert_to_tz(dt, pytz.timezone('US/Central'))
>>> print dt
2013-02-03 18:00:00-06:00

>>> # Convert aware CST time back to naive UTC time
>>> dt = fleming.convert_to_tz(dt, pytz.utc, return_naive=True)
>>> print dt
2013-02-04 00:00:00
```

4.2 add_timedelta

fleming.**add_timedelta**(*dt*, *td*, *within_tz=None*)

Adds a timedelta to a datetime. Can add timedeltas relative to a timezone.

Given a naive or aware datetime dt, add a timedelta td to it and return it. If within_tz is specified, the datetime arithmetic happens with regard to the timezone. Proper measures are used to ensure that datetime arithmetic across a DST border is handled properly.

Parameters

- **dt** (*datetime*) – A naive or aware datetime object. If it is naive, it is assumed to be UTC.
- **td** (*timedelta*) – A timedelta (or relativedelta) object to add to dt.
- **within_tz** (*pytz timezone*) – A pytz timezone object. If provided, dt will be converted to this timezone before datetime arithmetic and then converted back to its original timezone afterwards.

Return type datetime

Returns A datetime object that results from adding td to dt. The timezone of the returned datetime will be equivalent to the original timezone of dt (or its DST equivalent if a DST border was crossed). If the original time was naive, the returned value is naive.

```
>>> import pytz
>>> import datetime
>>> import fleming

>>> # Do a basic timedelta addition to a naive UTC time and create a naive UTC␣
↪time
>>> # two weeks in the future
>>> dt = datetime.datetime(2013, 3, 1)
>>> dt = fleming.add_timedelta(dt, datetime.timedelta(weeks=2))
>>> print dt
2013-03-15 00:00:00

>>> # Do addition on an EST datetime where the arithmetic does not cross over DST
>>> dt = fleming.convert_to_tz(dt, pytz.timezone('US/Eastern'))
>>> print dt
2013-03-14 20:00:00-04:00

>>> dt = fleming.add_timedelta(dt, datetime.timedelta(weeks=2, days=1))
>>> print dt
2013-03-29 20:00:00-04:00

>>> # Do timedelta arithmetic such that it starts in DST and crosses over into no␣
↪DST.
```

```
>>> # Note that the hours stay in tact and the timezone changes
>>> dt = fleming.add_timedelta(dt, datetime.timedelta(weeks=-4))
>>> print dt
2013-03-01 20:00:00-05:00

>>> # Take an aware UTC time and do datetime arithmetic in regards to EST. Do the␣
↪arithmetic
>>> # such that a DST border is crossed
>>> dt = datetime.datetime(2013, 3, 1, 5, tzinfo=pytz.utc)
>>> # It should be midnight in EST
>>> print fleming.convert_to_tz(dt, pytz.timezone('US/Eastern'))
2013-03-01 00:00:00-05:00

>>> # Do arithmetic on the UTC time with respect to EST.
>>> dt = fleming.add_timedelta(
...     dt, datetime.timedelta(weeks=2), within_tz=pytz.timezone('US/Eastern')
... )
...
>>> # The hour (4) of the returned UTC time is different that the original (5).
>>> print dt
2013-03-15 04:00:00+00:00

>>> # However, the hours in EST still reflect midnight
>>> print fleming.convert_to_tz(dt, pytz.timezone('US/Eastern'))
2013-03-15 00:00:00-04:00
```

4.3 floor

fleming.**floor**(*dt*, *within_tz=None*, *year=None*, *month=None*, *week=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *microsecond=None*, *extra_td_if_floor=None*)

Floors a datetime to the nearest time boundary.

Perform a floor on a datetime, rounding the datetime to its nearest provided interval. Available intervals are year, month, week, day, hour, minute, second, and microsecond.

For example, to round to the nearest month, provide month=1 as input. Give a value like 3 to the month argument and it will round to the nearest trimonth in a year (i.e. the nearest quarter). Values for other intervals operate in the same way, such as rounding down to the nearest day in a month (or nearest triday).

The only paramter to this function which is not like the others is the week parameter. The week parameter rounds the time down to its nearest Monday. In contrast to other intervals, week can only be 1 since it has no larger time interval from which to start a tri or quadweek for example.

Also, if the week parameter is provided, the year and month arguments are also not used in the floor calculation.

Note that multiple combinations of attributes can be used where they make sense, such as flooring to the nearest trimonth and triday (month=3, day=3).

Parameters

- **dt** (*datetime*) – A naive or aware datetime object. If it is naive, it is assumed to be UTC
- **within_tz** (*pytz timezone*) – A pytz timezone object. If given, the floor will be performed with respect to the timezone.
- **year** (*int*) – Specifies the yearly interval to round down to. Defaults to None.

- **month** (*int*) – Specifies the monthly interval (inside of a year) to round down to. Defaults to None.
- **week** (*int*) – Specifies to round to the beginning of the previous week. Defaults to None and only accepts a possible value of 1.
- **day** (*int*) – Specifies the daily interval to round down to (inside of a month). Defaults to None.
- **hour** (*int*) – Specifies the hourly interval to round down to (inside of a day). Defaults to None.
- **minute** (*int*) – Specifies the minute interval to round down to (inside of an hour). Defaults to None.
- **second** (*int*) – Specifies the second interval to round down to (inside of a minute). Defaults to None.
- **microsecond** (*int*) – Specifies the microsecond interval to round down to (inside of a second). Defaults to None.
- **extra_td_if_floor** (*timedelta*) – Only used by the ceil function. Specifies an extra timedelta to be added to the result if a floor has occurred.

Return type datetime

Returns A datetime object that results from flooring dt to the interval. The timezone of the returned datetime will be equivalent to the original timezone of dt (or its DST equivalent if a DST border was crossed). If the input time was naive, it returns a naive datetime object.

Raises ValueError if the interval is an invalid value.

```
>>> import datetime
>>> import pytz
>>> import fleming

>>> # Do basic floors in naive UTC time. Results are naive UTC.
>>> print fleming.floor(datetime.datetime(2013, 3, 3, 5), year=1)
2013-01-01 00:00:00

>>> print fleming.floor(datetime.datetime(2013, 3, 3, 5), month=1)
2013-03-01 00:00:00

>>> # Weeks start on Monday, so the floor will be for the previous Monday
>>> print fleming.floor(datetime.datetime(2013, 3, 3, 5), week=1)
2013-02-25 00:00:00

>>> print fleming.floor(datetime.datetime(2013, 3, 3, 5), day=1)
2013-03-03 00:00:00

>>> # Pass an aware datetime and return an aware datetime
>>> print fleming.floor(
...     datetime.datetime(2013, 3, 3, 5, tzinfo=pytz.utc), day=1)
2013-03-03 00:00:00+00:00

>>> # Perform a floor in EST. The result is in EST
>>> dt = fleming.convert_to_tz(
...    datetime.datetime(2013, 3, 4, 6), pytz.timezone('US/Eastern'))
>>> print dt
2013-03-04 01:00:00-05:00
```

```
>>> print fleming.floor(dt, year=1)
2013-01-01 00:00:00-05:00

>>> print fleming.floor(dt, day=1)
2013-03-04 00:00:00-05:00

>>> # Now perform a floor that starts out of DST and ends up in DST. The
>>> # timezones before and after the floor will be different, but the
>>> # time values are correct
>>> dt = fleming.convert_to_tz(
...     datetime.datetime(2013, 11, 28, 6), pytz.timezone('US/Eastern'))
...
>>> print dt
2013-11-28 01:00:00-05:00

>>> print fleming.floor(dt, month=1)
2013-11-01 00:00:00-04:00

>>> # Start with a naive UTC time and floor it with respect to EST
>>> dt = datetime.datetime(2013, 2, 1)
>>> # Since it is January 31 in EST, the resulting floored value
>>> # for a day will be the previous day. Also, the returned value is
>>> # in the original naive timezone of UTC
>>> print fleming.floor(dt, day=1, within_tz=pytz.timezone('US/Eastern'))
2013-01-31 00:00:00

>>> # Similarly, EST values can be floored relative to CST values.
>>> dt = fleming.convert_to_tz(
...     datetime.datetime(2013, 2, 1, 5), pytz.timezone('US/Eastern'))
>>> print dt
2013-02-01 00:00:00-05:00

>>> # Since it is January 31 in CST, the resulting floored value
>>> # for a day will be the previous day. Also, the returned value is
>>> # in the original timezone of EST
>>> print fleming.floor(dt, day=1, within_tz=pytz.timezone('US/Central'))
2013-01-31 00:00:00-05:00

>>> # Get the starting of a quarter by using month=3
>>> print fleming.floor(datetime.datetime(2013, 2, 4), month=3)
2013-01-01 00:00:00
```

4.4 ceil

fleming.**ceil**(*dt*, *within_tz=None*, *year=None*, *month=None*, *week=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *microsecond=None*)

Ceils a datetime to the nearest (above) time interval.

Perform a ceil on a datetime to the next closest interval in the future. For example, if month=1, this function will round up the time to the next month in the future. Larger numbers can be used, such as month=3, to round up to the beginning of the next quarter.

Note that this function allows combinations of interval variables (such as month=2 and day=2 to round up to the next duomonth of the year and next duoday of the month), but the smaller intervals are always not important since they will always be at the beginning of the larger interval.

Parameters

- **dt** (*datetime*) – A naive or aware datetime object. If it is naive, it is assumed to be UTC
- **within_tz** (*pytz timezone*) – A pytz timezone object. If given, the ceil will be performed with respect to the timezone.
- **year** (*int*) – Specifies the yearly interval to round up to. Defaults to None.
- **month** (*int*) – Specifies the monthly interval (inside of a year) to round up to. Defaults to None.
- **week** (*int*) – Specifies to round up to the beginning of the next week. Defaults to None and only accepts a possible value of 1.
- **day** (*int*) – Specifies the daily interval to round up to (inside of a month). Defaults to None.
- **hour** (*int*) – Specifies the hourly interval to round up to (inside of a day). Defaults to None.
- **minute** (*int*) – Specifies the minute interval to round up to (inside of an hour). Defaults to None.
- **second** (*int*) – Specifies the second interval to round up to (inside of a minute). Defaults to None.
- **microsecond** (*int*) – Specfies the microsecond interval to round up to (inside of a second). Defaults to None.

Returns A datetime object that results from ceiling dt to the next interval. The timezone of the returned datetime will be equivalent to the original timezone of dt (or its DST equivalent if a DST border was crossed). If the original datetime object was naive, the returned object is naive.

Raises ValueError if the interval is not a valid value.

```
>>> import datetime
>>> import pytz
>>> import fleming

>>> # Do basic ceils in naive UTC time. Results are naive UTC
>>> print fleming.ceil(datetime.datetime(2013, 3, 3, 5), year=1)
2014-01-01 00:00:00

>>> print fleming.ceil(datetime.datetime(2013, 3, 3, 5), month=1)
2013-04-01 00:00:00

>>> # Weeks start on Monday, so the floor will be for the next Monday
>>> print fleming.ceil(datetime.datetime(2013, 3, 3, 5), week=1)
2013-03-04 00:00:00

>>> print fleming.ceil(datetime.datetime(2013, 3, 3, 5), day=1)
2013-03-04 00:00:00

>>> # Use aware datetimes to return aware datetimes
>>> print fleming.ceil(
...     datetime.datetime(2013, 3, 3, 5, tzinfo=pytz.utc), day=1)
2013-03-04 00:00:00+00:00

>>> # Peform a ceil in CST. The result is in Pacfic time
>>> dt = fleming.convert_to_tz(
...     datetime.datetime(2013, 3, 4, 7), pytz.timezone('US/Pacific'))
```

```
>>> print dt
2013-03-03 23:00:00-08:00

>>> print fleming.ceil(dt, year=1)
2014-01-01 00:00:00-08:00

>>> print fleming.ceil(dt, day=1)
2013-03-04 00:00:00-08:00

>>> # Do a ceil with respect to EST. Since it is March 4 in EST, the
>>> # returned value is March 5 in Pacific time
>>> print fleming.ceil(dt, day=1, within_tz=pytz.timezone('US/Eastern'))
2013-03-05 00:00:00-08:00

>>> # Note that doing a ceiling on a time that is already on the boundary
>>> # returns the original time
>>> print fleming.ceil(datetime.datetime(2013, 4, 1), month=1)
2013-04-01 00:00:00
```

4.5 intervals

fleming.**intervals**(*start_dt*, *td*, *within_tz=None*, *stop_dt=None*, *is_stop_dt_inclusive=False*, *count=None*)

Returns a range of datetime objects with a timedelta interval.

Returns a range of datetime objects starting from start_dt and going in increments of timedelta td. If stop_dt is specified, the intervals go to stop_dt (and include stop_dt in the return if is_stop_dt_inclusive=True). If stop_dt is None, the count variable is used to control how many iterations are in the time intervals. If stop_dt is None and count is None, a generator will be returned that can yield any number of datetime objects.

Parameters

- **start_dt** (*datetime*) – A naive or aware datetime object from which to start the time intervals. If it is naive, it is assumed to be UTC.
- **td** (*timedelta*) – A timedelta object describing the time interval in the intervals.
- **within_tz** (*pytz timezone*) – A pytz timezone object. If provided, the intervals will be computed with respect to this timezone.
- **stop_dt** (*datetime*) – A naive or aware datetime object that specifies the end of the intervals. Defaults to being exclusive in the intervals. If naive, it is assumed to be in UTC.
- **is_stop_dt_inclusive** (*bool*) – True if the stop_dt should be included in the time intervals. Defaults to False.
- **count** (*int*) – If set, an integer specifying a count of intervals to use if stop_dt is None.

  If stop_dt is None and count is None, a generator will be returned that can yield any number of datetime objects.

Returns A generator of datetime objects. The datetime objects are in the original timezone of the start_dt (or its DST equivalent if a border is crossed). If the input is naive, the returned intervals are naive.

```
>>> import datetime
>>> import pytz
>>> import fleming
```

```
>>> # Using a naive UTC time, get intervals of time for every day.
>>> for dt in fleming.intervals(datetime.datetime(2013, 2, 3), datetime.
↪timedelta(days=1), count=5):
...     print dt
2013-02-03 00:00:00
2013-02-04 00:00:00
2013-02-05 00:00:00
2013-02-06 00:00:00
2013-02-07 00:00:00

>>> # Use an EST time. Do intervals of a day. Cross the DST time border on March␣
↪10th.
>>> est_dt = fleming.convert_to_tz(datetime.datetime(2013, 3, 9, 5), pytz.
↪timezone('US/Eastern'))
>>> for dt in fleming.intervals(est_dt, datetime.timedelta(days=1), count=5):
...     print dt
2013-03-09 00:00:00-05:00
2013-03-10 00:00:00-05:00
2013-03-11 00:00:00-04:00
2013-03-12 00:00:00-04:00
2013-03-13 00:00:00-04:00

>>> # Similarly, we can iterate through UTC times while doing the date range with␣
↪respect to EST. Note
>>> # that the UTC hour changes as the DST border is crossed on March 10th.
>>> for dt in fleming.intervals(
...         datetime.datetime(2013, 3, 9, 5),
...         datetime.timedelta(days=1),
...         within_tz=pytz.timezone('US/Eastern'),
...         count=5):
...     print dt
2013-03-09 05:00:00
2013-03-10 05:00:00
2013-03-11 04:00:00
2013-03-12 04:00:00
2013-03-13 04:00:00

>>> # If we don't specify a count or stop time, we can iterate indefinitely.
>>> for dt in fleming.intervals(datetime.datetime(2013, 1, 1), datetime.
↪timedelta(days=1)):
...     print dt
2013-01-01 00:00:00
2013-01-02 00:00:00
2013-01-03 00:00:00
....
2013-05-05 00:00:00
....
to the end of time...

>>> # Use a stop time. Note that the stop time is exclusive
>>> for dt in fleming.intervals(
...         datetime.datetime(2013, 3, 9),
...         datetime.timedelta(weeks=1),
...         stop_dt=datetime.datetime(2013, 3, 23)):
...     print dt
2013-03-09 00:00:00
2013-03-16 00:00:00
```

```
>>> # Make the previous range inclusive
>>> for dt in fleming.intervals(
...         datetime.datetime(2013, 3, 9), datetime.timedelta(weeks=1), stop_
↪dt=datetime.datetime(2013, 3, 23),
...         is_stop_dt_inclusive=True):
...     print dt
2013-03-09 00:00:00
2013-03-16 00:00:00
2013-03-23 00:00:00

>>> # Arbitrary timedeltas can be used for any sort of time range
>>> for dt in fleming.intervals(
...         datetime.datetime(2013, 3, 9), datetime.timedelta(days=1, hours=2,␣
↪minutes=1), count=5):
...     print dt
2013-03-09 00:00:00
2013-03-10 02:01:00
2013-03-11 04:02:00
2013-03-12 06:03:00
2013-03-13 08:04:00
```

4.6 unix_time

fleming.**unix_time**(*dt*, *within_tz=None*, *return_ms=False*)

Converts a datetime object to a unix timestamp.

Converts a naive or aware datetime object to unix timestamp. If within_tz is present, the timestamp returned is relative to that time zone.

Parameters

- **dt** (*datetime*) – A naive or aware datetime object. If it is naive, it is assumed to be UTC.
- **within_tz** (*pytz timezone*) – A pytz timezone object if the user wishes to return the unix time relative to another timezone.
- **return_ms** (*bool*) – A boolean specifying to return the value in milliseconds since the Unix epoch. Defaults to False.

Return type int or float

Returns An integer timestamp since the Unix epoch. If return_ms is True, returns the timestamp in milliseconds.

```
>>> import datetime
>>> import pytz
>>> import fleming

>>> # Do a basic naive UTC conversion
>>> dt = datetime.datetime(2013, 4, 2)
>>> print fleming.unix_time(dt)
1364860800

>>> # Convert a time in a different timezone
>>> dt = fleming.convert_to_tz(
...     datetime.datetime(2013, 4, 2, 4), pytz.timezone('US/Eastern'))
```

```
>>> print dt
2013-04-02 00:00:00-04:00

>>> print fleming.unix_time(dt)
1364875200

>>> # Print millisecond returns
>>> print fleming.unix_time(dt, return_ms=True)
1364875200000

>>> # Do a unix_time conversion with respect to another timezone. When
>>> # it is converted back to a datetime, the time values are correct.
>>> # The original timezone, however, needs to be added back
>>> dt = datetime.datetime(2013, 2, 1, 5)

>>> # Print its EST time for later reference
>>> print fleming.convert_to_tz(dt, pytz.timezone('US/Eastern'))
2013-02-01 00:00:00-05:00

>>> unix_tz_dt = fleming.unix_time(
...     dt, within_tz=pytz.timezone('US/Eastern'))
>>> print unix_tz_dt
1359676800

>>> # The datetime should match the original UTC time converted in
>>> # the timezone of the within_tz parameter. Tz information is
>>> # originally lost when converting to unix time, so replace the
>>> # tzinfo object here
>>> dt = datetime.datetime.fromtimestamp(unix_tz_dt).replace(
...     tzinfo=pytz.timezone('US/Eastern'))
>>> print dt
2013-02-01 00:00:00-05:00
```

CHAPTER 5

Contributing

Contributions and issues are most welcome! All issues and pull requests are handled through github on the ambitioninc repository. Please check for any existing issues before filing a new one!

5.1 Running the tests

To cloned the source code, you can run the code quality and unit tests by running:

```
$ git clone git://github.com/ambitioninc/fleming.git
$ cd fleming
$ virtualenv env
$ . env/bin/activate
$ pip install nose>=1.3.0
$ python setup.py install
$ python setup.py nosetests
```

While 100% code coverage does not make a library bug-free, it significantly reduces the number of easily caught bugs! Please make sure coverage is at 100% before submitting a pull request!

5.2 Code Quality

For code quality, please run:

```
$ pip install flake8
$ flake8 .
```

5.3 Code Styling

Please arrange imports with the following style

```
# Standard library imports
import os

# Third party package imports
from mock import patch

# Local package imports
from fleming import floor
```

Please follow Google's python style guide wherever possible.

5.4 Building the docs

When in the project directory:

```
$ pip install -r requirements/docs.txt
$ python setup.py build_sphinx
$ open docs/_build/html/index.html
```

5.5 Release Checklist

Before a new release, please go through the following checklist:

- Bump version in fleming/version.py
- Add a release note in docs/release_notes.rst
- Git tag the version
- Upload to pypi:

  ```
  pip install wheel
  python setup.py sdist bdist_wheel upload
  ```

5.6 Vulnerability Reporting

For any security issues, please do NOT file an issue or pull request on github! Please contact security@ambition.com with the GPG key provided on Ambition's website.

CHAPTER 6

Release Notes

6.1 v0.4.6

- Officially support python 3.6

6.2 v0.4.5

- Fixed a bug that cropped up when trying to floor times to quarter boundaries
- Fixed a bug that caused tests to fail on machines having non-UTC timezones

6.3 v0.4.4

- Officially support python 3.5

6.4 v0.4.1

- Added sphinx documentation and converted all docs to RestructuredText

CHAPTER 7

License

Fleming is licensed under the MIT License:

> Copyright (c) 2014: Ambition Solutions Inc.
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

External libraries, if used, include their own licenses:

- pytz

Index

A

add_timedelta() (in module fleming), 8

C

ceil() (in module fleming), 11
convert_to_tz() (in module fleming), 7

F

floor() (in module fleming), 9

I

intervals() (in module fleming), 13

U

unix_time() (in module fleming), 15