dateutil Documentation

Release 2.8.1

dateutil

Nov 03, 2019

Contents

The *dateutil* module provides powerful extensions to the standard *datetime* module, available in Python.

CHAPTER 1

Installation

dateutil can be installed from PyPI using *pip* (note that the package name is different from the importable name):

```
pip install python-dateutil
```

CHAPTER 2

Download

dateutil is available on PyPI https://pypi.org/project/python-dateutil/

The documentation is hosted at: https://dateutil.readthedocs.io/en/stable/

CHAPTER 3

Code

The code and issue tracker are hosted on GitHub: https://github.com/dateutil/dateutil/

CHAPTER 4

Features

- Computing of relative deltas (next month, next year, next Monday, last week of month, etc);
- Computing of relative deltas between two given date and/or datetime objects;
- Computing of dates based on very flexible recurrence rules, using a superset of the iCalendar specification. Parsing of RFC strings is supported as well.
- Generic parsing of dates in almost any string format;
- Timezone (tzinfo) implementations for tzfile(5) format files (/etc/localtime, /usr/share/zoneinfo, etc), TZ environment string (in all known formats), iCalendar format files, given ranges (with help from relative deltas), local machine timezone, fixed offset timezone, UTC timezone, and Windows registry-based time zones.
- Internal up-to-date world timezone information based on Olson's database.
- Computing of Easter Sunday dates for any given year, using Western, Orthodox or Julian algorithms;
- A comprehensive test suite.

CHAPTER 5

Quick example

Here's a snapshot, just to give an idea about the power of the package. For more examples, look at the documentation.

Suppose you want to know how much time is left, in years/months/days/etc, before the next easter happening on a year with a Friday 13th in August, and you want to get today's date out of the "date" unix system command. Here is the code:

```
>>> from dateutil.relativedelta import *
>>> from dateutil.easter import *
>>> from dateutil.rrule import *
>>> from dateutil.parser import *
>>> from datetime import *
>>> now = parse("Sat Oct 11 17:13:46 UTC 2003")
>>> today = now.date()
>>> year = rrule(YEARLY,dtstart=now,bymonth=8,bymonthday=13,byweekday=FR)[0].year
>>> rdelta = relativedelta(easter(year), today)
>>> print("Today is: %s" % today)
Today is: 2003-10-11
>>> print("Year with next Aug 13th on a Friday is: %s" % year)
Year with next Aug 13th on a Friday is: 2004
>>> print("How far is the Easter of that year: %s" % rdelta)
How far is the Easter of that year: relativedelta(months=+6)
>>> print("And the Easter of that year is: %s" % (today+rdelta))
And the Easter of that year is: 2004-04-11
```

Being exactly 6 months ahead was **really** a coincidence :)

CHAPTER 6

Contributing

We welcome many types of contributions - bug reports, pull requests (code, infrastructure or documentation fixes). For more information about how to contribute to the project, see the `CONTRIBUTING.md` file in the repository.

CHAPTER 7

Author

The dateutil module was written by Gustavo Niemeyer <gustavo@niemeyer.net> in 2003.

It is maintained by:

- Gustavo Niemeyer <gustavo@niemeyer.net> 2003-2011
- Tomi Pieviläinen <tomi.pievilainen@iki.fi> 2012-2014
- Yaron de Leeuw <me@jarondl.net> 2014-2016
- Paul Ganssle <paul@ganssle.io> 2015-

Starting with version 2.4.1, all source and binary distributions will be signed by a PGP key that has, at the very least, been signed by the key which made the previous release. A table of release signing keys can be found below:

Releases	Signing key fingerprint
2.4.1-	6B49 ACBA DCF6 BD1C A206 67AB CD54 FCE3 D964 BEFB (mirror)

CHAPTER 8

Contact

Our mailing list is available at dateutil@python.org. As it is hosted by the PSF, it is subject to the PSF code of conduct.

CHAPTER 9

License

All contributions after December 1, 2017 released under dual license - either Apache 2.0 License or the BSD 3-Clause License. Contributions before December 1, 2017 - except those those explicitly relicensed - are released only under the BSD 3-Clause License.

CHAPTER 10

Documentation

Contents:

10.1 Changelog

10.1.1 Version 2.8.1 (2019-11-03)

Data updates

- Updated tzdata version to 2019c.

Bugfixes

- Fixed a race condition in the `tzoffset` and `tzstr` "strong" caches on Python 2.7. Reported by @kainjow (gh issue #901).
- Parsing errors will now raise `ParserError`, a subclass of `ValueError`, which has a nicer string representation. Patch by @gfyoung (gh pr #881).
- `parser.parse` will now raise `TypeError` when `tzinfos` is passed a type that cannot be interpreted as a time zone. Prior to this change, it would raise an `UnboundLocalError` instead. Patch by @jbrockmendel (gh pr #891).
- Changed error message raised when when passing a `bytes` object as the time zone name to gettz in Python 3. Reported and fixed by @labrys () (gh issue #927, gh pr #935).
- Changed compatibility logic to support a potential Python 4.0 release. Patch by Hugo van Kemenade (gh pr #950).
- Updated many modules to use `tz.UTC` in favor of `tz.tzutc()` internally, to avoid an unnecessary function call. (gh pr #910).

- Fixed issue where `dateutil.tz` was using a backported version of `contextlib.nullcontext` even in Python 3.7 due to a malformed import statement. (gh pr #963).

Tests

- Switched from using assertWarns to using pytest.warns in the test suite. (gh pr #969).
- Fix typo in setup.cfg causing PendingDeprecationWarning to not be explicitly specified as an error in the warnings filter. (gh pr #966)
- Fixed issue where `test_tzlocal_offset_equal` would fail in certain environments (such as FreeBSD) due to an invalid assumption about what time zone names are provided. Reported and fixed by Kubilay Kocak (gh issue #918, pr #928).
- Fixed a minor bug in `test_isoparser` related to `bytes`/`str` handling. Fixed by @fhuang5 (gh issue #776, gh pr #879).
- Explicitly listed all markers used in the pytest configuration. (gh pr #915)
- Extensive improvements to the parser test suite, including the adoption of `pytest`-style tests and the addition of parametrization of several test cases. Patches by @jbrockmendel (gh prs #735, #890, #892, #894).
- Added tests for tzinfos input types. Patch by @jbrockmendel (gh pr #891).
- Fixed failure of test suite when changing the TZ variable is forbidden. Patch by @shadchin (gh pr #893).
- Pinned all test dependencies on Python 3.3. (gh prs #934, #962)

Documentation changes

- Fixed many misspellings, typos and styling errors in the comments and documentation. Patch by Hugo van Kemenade (gh pr #952).

Misc

- Added Python 3.8 to the trove classifiers. (gh pr #970)
- Moved as many keys from `setup.py` to `setup.cfg` as possible. Fixed by @FakeNameSE, @aquinlan82, @jachen20, and @gurgenz221 (gh issue #871, gh pr #880).
- Reorganized `parser` methods by functionality. Patch by @jbrockmendel (gh pr #882).
- Switched `release.py` over to using `pep517.build` for creating releases, rather than direct invocations of `setup.py`. Fixed by @smeng10 (gh issue #869, gh pr #875).
- Added a "build" environment into the tox configuration, to handle dependency management when making releases. Fixed by @smeng10 (gh issue #870,r gh pr #876).
- GH #916, GH #971

10.1.2 Version 2.8.0 (2019-02-04)

Data updates

- Updated tzdata version to to 2018i.

Features

- Added support for `EXDATE` parameters when parsing `rrule` strings. Reported by @mlorant (gh issue #410), fixed by @nicoe (gh pr #859).
- Added support for sub-minute time zone offsets in Python 3.6+. Fixed by @cssherry (gh issue #582, pr #763)
- Switched the `tzoffset`, `tzstr` and `gettz` caches over to using weak references, so that the cache expires when no other references to the original `tzinfo` objects exist. This cache-expiry behavior is not guaranteed in the public interface and may change in the future. To improve performance in the case where transient references to the same time zones are repeatedly created but no strong reference is continuously held, a smaller "strong value" cache was also added. Weak value cache implemented by @cs-cordero (gh pr #672, #801), strong cache added by Gökçen Nurlu (gh issue #691, gh pr #761)

Bugfixes

- Add support for ISO 8601 times with comma as the decimal separator in the `dateutil.parser.isoparse` function. (gh pr #721)
- Changed handling of `T24:00` to be compliant with the standard. `T24:00` now represents midnight on the *following* day. Fixed by @cheukting (gh issue #658, gh pr #751)
- Fixed an issue where `isoparser.parse_isotime` was unable to handle the `24:00` variant representation of midnight. (gh pr #773)
- Added support for more than 6 fractional digits in *isoparse*. Reported and fixed by @jayschwa (gh issue #786, gh pr #787).
- Added 'z' (lower case Z) as valid UTC time zone in isoparser. Reported by @cjgibson (gh issue #820). Fixed by @Cheukting (gh pr #822)
- Fixed a bug with base offset changes during DST in `tzfile`, and refactored the way base offset changes are detected. Originally reported on Stack Overflow by @MartinThoma. (gh issue #812, gh pr #810)
- Fixed error condition in `tz.gettz` when a non-ASCII timezone is passed on Windows in Python 2.7. (gh issue #802, pr #861)
- Improved performance and inspection properties of `tzname` methods. (gh pr #811)
- Removed unnecessary binary_type compatibility shims. Added by @jdufresne (gh pr #817)
- Changed `python setup.py test` to print an error to `stderr` and exit with 1 instead of 0. Reported and fixed by @hroncok (gh pr #814)
- Added a `pyproject.toml` file with build requirements and an explicitly specified build backend. (gh issue #736, gh prs #746, #863)

Documentation changes

- Added documentation for the `rrule.rrulestr` function. Fixed by @prdickson (gh issue #623, gh pr #762)
- Add documentation for the `dateutil.tz.win` module and mocked out certain Windows-specific modules so that autodoc can still be run on non-Windows systems. (gh issue #442, pr #715)
- Added changelog to documentation. (gh issue #692, gh pr #707)
- Improved documentation on the use of `until` and `count` parameters in `rrule`. Fixed by @lucaferocino (gh pr #755).
- Added an example of how to use a custom `parserinfo` subclass to parse non-standard datetime formats in the examples documentation for `parser`. Added by @prdickson (gh #753)

- Expanded the description and examples in the `relativedelta` class. Contributed by @andrewcbennett (gh pr #759)
- Improved the contributing documentation to clarify where to put new changelog files. Contributed by @andrewcbennett (gh pr #757)
- Fixed a broken doctest in the `relativedelta` module. Fixed by @nherriot (gh pr #758).
- Reorganized `dateutil.tz` documentation and fixed issue with the `dateutil.tz` docstring. (gh pr #714)

Misc

- GH #720, GH #723, GH #726, GH #727, GH #740, GH #750, GH #760, GH #767, GH #772, GH #773, GH #780, GH #784, GH #785, GH #791, GH #799, GH #813, GH #836, GH #839, GH #857

10.1.3 Version 2.7.5 (2018-10-27)

Data updates

- Update tzdata to 2018g

10.1.4 Version 2.7.4 (2018-10-24)

Data updates

- Updated tzdata version to 2018f.

10.1.5 Version 2.7.3 (2018-05-09)

Data updates

- Update tzdata to 2018e. (gh pr #710)

Bugfixes

- Fixed an issue where `parser.parse` would raise `Decimal`-specific errors instead of a standard `ValueError` if certain malformed values were parsed (e.g. `NaN` or infinite values). Reported and fixed by @amureki (gh issue #662, gh pr #679).
- Fixed issue in `parser` where a `tzinfos` call explicitly returning `None` would throw a `ValueError`. Fixed by @parsethis (gh issue #661, gh pr #681)
- Fixed incorrect parsing of certain dates earlier than 100 AD when represented in the form "%B.%Y.%d", e.g. "December.0031.30". (gh issue #687, pr #700)
- Added time zone inference when initializing an `rrule` with a specified `UNTIL` but without an explicitly specified `DTSTART`; the time zone of the generated `DTSTART` will now be taken from the `UNTIL` rule. Reported by @href (gh issue #652). Fixed by @absreim (gh pr #693).

Documentation changes

- Corrected link syntax and updated URL to https for ISO year week number notation in relativedelta examples. (gh issue #670, pr #711)
- Add doctest examples to tzfile documentation. Done by @weatherpattern and @pganssle (gh pr #671)
- Updated the documentation for relativedelta. Removed references to tuple arguments for weekday, explained effect of weekday(_, 1) and better explained the order of operations that relativedelta applies. Fixed by @kvn219 @huangy22 and @ElliotJH (gh pr #673)
- Added changelog to documentation. (gh issue #692, gh pr #707)
- Changed order of keywords in rrule docstring. Reported and fixed by @rmahajan14 (gh issue #686, gh pr #695).
- Added documentation for `dateutil.tz.gettz`. Reported by @pganssle (gh issue #647). Fixed by @weatherpattern (gh pr #704)
- Cleaned up malformed RST in the `tz` documentation. (gh issue #702, gh pr #706)
- Changed the default theme to `sphinx_rtd_theme`, and changed the sphinx configuration accordingly. (gh pr #707)
- Reorganized `dateutil.tz` documentation and fixed issue with the `dateutil.tz` docstring. (gh pr #714)

Misc

- GH #674, GH #688, GH #699

10.1.6 Version 2.7.2 (2018-03-26)

Bugfixes

- Fixed an issue with the setup script running in non-UTF-8 environment. Reported and fixed by @gergondet (gh pr #651)

Misc

- GH #655

10.1.7 Version 2.7.1 (2018-03-24)

Data updates

- Updated tzdata version to 2018d.

Bugfixes

- Fixed issue where parser.parse would occasionally raise decimal.Decimal-specific error types rather than ValueError. Reported by @amureki (gh issue #632). Fixed by @pganssle (gh pr #636).
- Improve error message when rrule's dtstart and until are not both naive or both aware. Reported and fixed by @ryanpetrello (gh issue #633, gh pr #634)

Misc

- GH #644, GH #648

10.1.8 Version 2.7.0

- Dropped support for Python 2.6 (gh pr #362 by @jdufresne)
- Dropped support for Python 3.2 (gh pr #626)
- Updated zoneinfo file to 2018c (gh pr #616)
- Changed licensing scheme so all new contributions are dual licensed under Apache 2.0 and BSD. (gh pr #542, issue #496)
- Added __all__ variable to the root package. Reported by @tebriel (gh issue #406), fixed by @mariocj89 (gh pr #494)
- Added python_requires to setup.py so that pip will distribute the right version of dateutil. Fixed by @jakec-github (gh issue #537, pr #552)
- Added the utils submodule, for miscellaneous utilities.
- Added within_delta function to utils - added by @justanr (gh issue #432, gh pr #437)
- Added today function to utils (gh pr #474)
- Added default_tzinfo function to utils (gh pr #475), solving an issue reported by @nealmcb (gh issue #94)
- Added dedicated ISO 8601 parsing function isoparse (gh issue #424). Initial implementation by @pganssle in gh pr #489 and #622, with a pre-release fix by @kirit93 (gh issue #546, gh pr #573).
- Moved parser module into parser/_parser.py and officially deprecated the use of several private functions and classes from that module. (gh pr #501, #515)
- Tweaked parser error message to include rejected string format, added by @pbiering (gh pr #300)
- Add support for parsing bytesarray, reported by @uckelman (gh issue #417) and fixed by @uckelman and @pganssle (gh pr #514)
- Started raising a warning when the parser finds a timezone string that it cannot construct a tzinfo instance for (rather than succeeding with no indication of an error). Reported and fixed by @jbrockmendel (gh pr #540)
- Dropped the use of assert in the parser. Fixed by @jbrockmendel (gh pr #502)
- Fixed to assertion logic in parser to support dates like '2015-15-May', reported and fixed by @jbrockmendel (gh pr #409)
- Fixed IndexError in parser on dates with trailing colons, reported and fixed by @jbrockmendel (gh pr #420)
- Fixed bug where hours were not validated, leading to improper parse. Reported by @heappro (gh pr #353), fixed by @jbrockmendel (gh pr #482)
- Fixed problem parsing strings in %b-%Y-%d format. Reported and fixed by @jbrockmendel (gh pr #481)
- Fixed problem parsing strings in the %d%B%y format. Reported by @asishm (gh issue #360), fixed by @jbrockmendel (gh pr #483)
- Fixed problem parsing certain unambiguous strings when year <99 (gh pr #510). Reported by @alexwlchan (gh issue #293).
- Fixed issue with parsing an unambiguous string representation of an ambiguous datetime such that if possible the correct value for fold is set. Fixes issue reported by @JordonPhillips and @pganssle (gh issue #318, #320, gh pr #517)

- Fixed issue with improper rounding of fractional components. Reported by @dddmello (gh issue #427), fixed by @m-dz (gh pr #570)
- Performance improvement to parser from removing certain min() calls. Reported and fixed by @jbrockmendel (gh pr #589)
- Significantly refactored parser code by @jbrockmendel (gh prs #419, #436, #490, #498, #539) and @pganssle (gh prs #435, #468)
- Implemented of __hash__ for relativedelta and weekday, reported and fixed by @mrigor (gh pr #389)
- Implemented __abs__ for relativedelta. Reported by @binnisb and @pferreir (gh issue #350, pr #472)
- Fixed relativedelta.weeks property getter and setter to work for both negative and positive values. Reported and fixed by @souliane (gh issue #459, pr #460)
- Fixed issue where passing whole number floats to the months or years arguments of the relativedelta constructor would lead to errors during addition. Reported by @arouanet (gh pr #411), fixed by @lkollar (gh pr #553)
- Added a pre-built tz.UTC object representing UTC (gh pr #497)
- Added a cache to tz.gettz so that by default it will return the same object for identical inputs. This will change the semantics of certain operations between datetimes constructed with tzinfo=tz.gettz(...). (gh pr #628)
- Changed the behavior of tz.tzutc to return a singleton (gh pr #497, #504)
- Changed the behavior of tz.tzoffset to return the same object when passed the same inputs, with a corresponding performance improvement (gh pr #504)
- Changed the behavior of tz.tzstr to return the same object when passed the same inputs. (gh pr #628)
- Added .instance alternate constructors for tz.tzoffset and tz.tzstr, to allow the construction of a new instance if desired. (gh pr #628)
- Added the tz.gettz.nocache function to allow explicit retrieval of a new instance of the relevant tzinfo. (gh pr #628)
- Expand definition of tz.tzlocal equality so that the local zone is allow equality with tzoffset and tzutc. (gh pr #598)
- Deprecated the idiosyncratic tzstr format mentioned in several examples but evidently designed exclusively for dateutil, and very likely not used by any current users. (gh issue #595, gh pr #606)
- Added the tz.resolve_imaginary function, which generates a real date from an imaginary one, if necessary. Implemented by @Cheukting (gh issue #339, gh pr #607)
- Fixed issue where the tz.tzstr constructor would erroneously succeed if passed an invalid value for tzstr. Fixed by @pablogsal (gh issue #259, gh pr #581)
- Fixed issue with tz.gettz for TZ variables that start with a colon. Reported and fixed by @lapointexavier (gh pr #601)
- Added a lock to tz.tzical's cache. Reported and fixed by @Unrud (gh pr #430)
- Fixed an issue with fold support on certain Python 3 implementations that used the pre-3.6 pure Python implementation of datetime.replace, most notably pypy3 (gh pr #446).
- Added support for VALUE=DATE-TIME for DTSTART in rrulestr. Reported by @potuz (gh issue #401) and fixed by @Unrud (gh pr #429)
- Started enforcing that within VTIMEZONE, the VALUE parameter can only be omitted or DATE-TIME, per RFC 5545. Reported by @Unrud (gh pr #439)
- Added support for TZID parameter for DTSTART in rrulestr. Reported and fixed by @ryanpetrello (gh issue #614, gh pr #624)

- Added 'RRULE:' prefix to rrule strings generated by rrule.__str__, in compliance with the RFC. Reported by @AndrewPashkin (gh issue #86), fixed by @jarondl and @mlorant (gh pr #450)
- Switched to setuptools_scm for version management, automatically calculating a version number from the git metadata. Reported by @jreback (gh issue #511), implemented by @Sulley38 (gh pr #564)
- Switched setup.py to use find_packages, and started testing against pip installed versions of dateutil in CI. Fixed issue with parser import discovered by @jreback in pandas-dev/pandas#18141. (gh issue #507, pr #509)
- Switched test suite to using pytest (gh pr #495)
- Switched CI over to use tox. Fixed by @gaborbernat (gh pr #549)
- Added a test-only dependency on freezegun. (gh pr #474)
- Reduced number of CI builds on Appveyor. Fixed by @kirit93 (gh issue #529, gh pr #579)
- Made xfails strict by default, so that an xpass is a failure. (gh pr #567)
- Added a documentation generation stage to tox and CI. (gh pr #568)
- Added an explicit warning when running python setup.py explaining how to run the test suites with pytest. Fixed by @lkollar. (gh issue #544, gh pr #548)
- Added requirements-dev.txt for test dependency management (gh pr #499, #516)
- Fixed code coverage metrics to account for Windows builds (gh pr #526)
- Fixed code coverage metrics to NOT count xfails. Fixed by @gaborbernat (gh issue #519, gh pr #563)
- Style improvement to zoneinfo.tzfile that was confusing to static type checkers. Reported and fixed by @quodlibetor (gh pr #485)
- Several unused imports were removed by @jdufresne. (gh pr #486)
- Switched `isinstance(*, collections.Callable)` to callable, which is available on all supported Python versions. Implemented by @jdufresne (gh pr #612)
- Added CONTRIBUTING.md (gh pr #533)
- Added AUTHORS.md (gh pr #542)
- Corrected setup.py metadata to reflect author vs. maintainer, (gh issue #477, gh pr #538)
- Corrected README to reflect that tests are now run in pytest. Reported and fixed by @m-dz (gh issue #556, gh pr #557)
- Updated all references to RFC 2445 (iCalendar) to point to RFC 5545. Fixed by @mariocj89 (gh issue #543, gh pr #555)
- Corrected parse documentation to reflect proper integer offset units, reported and fixed by @abrugh (gh pr #458)
- Fixed dangling parenthesis in tzoffset documentation (gh pr #461)
- Started including the license file in wheels. Reported and fixed by @jdufresne (gh pr #476)
- Indentation fixes to parser docstring by @jbrockmendel (gh pr #492)
- Moved many examples from the "examples" documentation into their appropriate module documentation pages. Fixed by @Tomasz-Kluczkowski and @jakec-github (gh pr #558, #561)
- Fixed documentation so that the parser.isoparse documentation displays. Fixed by @alexchamberlain (gh issue #545, gh pr #560)
- Refactored build and release sections and added setup instructions to CONTRIBUTING. Reported and fixed by @kynan (gh pr #562)
- Cleaned up various dead links in the documentation. (gh pr #602, #608, #618)

10.1.9 Version 2.6.1

- Updated zoneinfo file to 2017b. (gh pr #395)
- Added Python 3.6 to CI testing (gh pr #365)
- Removed duplicate test name that was preventing a test from being run. Reported and fixed by @jdufresne (gh pr #371)
- Fixed testing of folds and gaps, particularly on Windows (gh pr #392)
- Fixed deprecated escape characters in regular expressions. Reported by @nascheme and @thierryba (gh issue #361), fixed by @thierryba (gh pr #358)
- Many PEP8 style violations and other code smells were fixed by @jdufresne (gh prs #358, #363, #364, #366, #367, #368, #372, #374, #379, #380, #398)
- Improved performance of tzutc and tzoffset objects. (gh pr #391)
- Fixed issue with several time zone classes around DST transitions in any zones with +0 standard offset (e.g. Europe/London) (gh issue #321, pr #390)
- Fixed issue with fuzzy parsing where tokens similar to AM/PM that are in the end skipped were dropped in the fuzzy_with_tokens list. Reported and fixed by @jbrockmendel (gh pr #332).
- Fixed issue with parsing dates of the form X m YY. Reported by @jbrockmendel. (gh issue #333, pr #393)
- Added support for parser weekdays with less than 3 characters. Reported by @arcadefoam (gh issue #343), fixed by @jonemo (gh pr #382)
- Fixed issue with the addition and subtraction of certain relativedeltas. Reported and fixed by @kootenpv (gh issue #346, pr #347)
- Fixed issue where the COUNT parameter of rrules was ignored if 0. Fixed by @mshenfield (gh pr #330), reported by @vaultah (gh issue #329).
- Updated documentation to include the new tz methods. (gh pr #324)
- Update documentation to reflect that the parser can raise TypeError, reported and fixed by @tomchuk (gh issue #336, pr #337)
- Fixed an incorrect year in a parser doctest. Fixed by @xlotlu (gh pr #357)
- Moved version information into _version.py and set up the versions more granularly.

10.1.10 Version 2.6.0

- Added PEP-495-compatible methods to address ambiguous and imaginary dates in time zones in a backwards-compatible way. Ambiguous dates and times can now be safely represented by all dateutil time zones. Many thanks to Alexander Belopolski (@abalkin) and Tim Peters @tim-one for their inputs on how to address this. Original issues reported by Yupeng and @zed (lP: 1390262, gh issues #57, #112, #249, #284, #286, prs #127, #225, #248, #264, #302).
- Added new methods for working with ambiguous and imaginary dates to the tz module. datetime_ambiguous() determines if a datetime is ambiguous for a given zone and datetime_exists() determines if a datetime exists in a given zone. This works for all fold-aware datetimes, not just those provided by dateutil. (gh issue #253, gh pr #302)
- Fixed an issue where dst() in Portugal in 1996 was returning the wrong value in tz.tzfile objects. Reported by @abalkin (gh issue #128, pr #225)
- Fixed an issue where zoneinfo.ZoneInfoFile errors were not being properly deep-copied. (gh issue #226, pr #225)

- Refactored tzwin and tzrange as a subclass of a common class, tzrangebase, as there was substantial overlapping functionality. As part of this change, tzrange and tzstr now expose a transitions() function, which returns the DST on and off transitions for a given year. (gh issue #260, pr #302)
- Deprecated zoneinfo.gettz() due to confusion with tz.gettz(), in favor of get() method of zoneinfo.ZoneInfoFile objects. (gh issue #11, pr #310)
- For non-character, non-stream arguments, parser.parse now raises TypeError instead of AttributeError. (gh issues #171, #269, pr #247)
- Fixed an issue where tzfile objects were not properly handling dst() and tzname() when attached to datetime.time objects. Reported by @ovacephaloid. (gh issue #292, pr #309)
- /usr/share/lib/zoneinfo was added to TZPATHS for compatibility with Solaris systems. Reported by @dhduvall (gh issue #276, pr #307)
- tzoffset and tzrange objects now accept either a number of seconds or a datetime.timedelta() object wherever previously only a number of seconds was allowed. (gh pr #264, #277)
- datetime.timedelta objects can now be added to relativedelta objects. Reported and added by Alec Nikolas Reiter (@justanr) (gh issue #282, pr #283
- Refactored relativedelta.weekday and rrule.weekday into a common base class to reduce code duplication. (gh issue #140, pr #311)
- An issue where the WKST parameter was improperly rendering in str(rrule) was reported and fixed by Daniel LePage (@dplepage). (gh issue #262, pr #263)
- A replace() method has been added to rrule objects by @jendas1, which creates new rrule with modified attributes, analogous to datetime.replace (gh pr #167)
- Made some significant performance improvements to rrule objects in Python 2.x (gh pr #245)
- All classes defining equality functions now return NotImplemented when compared to unsupported classes, rather than raising TypeError, to allow other classes to provide fallback support. (gh pr #236)
- Several classes have been marked as explicitly unhashable to maintain identical behavior between Python 2 and 3. Submitted by Roy Williams (@rowillia) (gh pr #296)
- Trailing whitespace in easter.py has been removed. Submitted by @OmgImAlexis (gh pr #299)
- Windows-only batch files in build scripts had line endings switched to CRLF. (gh pr #237)
- @adamchainz updated the documentation links to reflect that the canonical location for readthedocs links is now at .io, not .org. (gh pr #272)
- Made some changes to the CI and codecov to test against newer versions of Python and pypy, and to adjust the code coverage requirements. For the moment, full pypy3 compatibility is not supported until a new release is available, due to upstream bugs in the old version affecting PEP-495 support. (gh prs #265, #266, #304, #308)
- The full PGP signing key fingerprint was added to the README.md in favor of the previously used long-id. Reported by @valholl (gh issue #287, pr #304)
- Updated zoneinfo to 2016i. (gh issue #298, gh pr #306)

10.1.11 Version 2.5.3

- Updated zoneinfo to 2016d
- Fixed parser bug where unambiguous datetimes fail to parse when dayfirst is set to true. (gh issue #233, pr #234)
- Bug in zoneinfo file on platforms such as Google App Engine which do not do not allow importing of subprocess.check_call was reported and fixed by @savraj (gh issue #239, gh pr #240)

- Fixed incorrect version in documentation (gh issue #235, pr #243)

10.1.12 Version 2.5.2

- Updated zoneinfo to 2016c
- Fixed parser bug where yearfirst and dayfirst parameters were not being respected when no separator was present. (gh issue #81 and #217, pr #229)

10.1.13 Version 2.5.1

- Updated zoneinfo to 2016b
- Changed MANIFEST.in to explicitly include test suite in source distributions, with help from @koobs (gh issue #193, pr #194, #201, #221)
- Explicitly set all line-endings to LF, except for the NEWS file, on a per-repository basis (gh pr #218)
- Fixed an issue with improper caching behavior in rruleset objects (gh issue #104, pr #207)
- Changed to an explicit error when rrulestr strings contain a missing BYDAY (gh issue #162, pr #211)
- tzfile now correctly handles files containing leapcnt (although the leapcnt information is not actually used). Contributed by @hjoukl (gh issue #146, pr #147)
- Fixed recursive import issue with tz module (gh pr #204)
- Added compatibility between tzwin objects and datetime.time objects (gh issue #216, gh pr #219)
- Refactored monolithic test suite by module (gh issue #61, pr #200 and #206)
- Improved test coverage in the relativedelta module (gh pr #215)
- Adjusted documentation to reflect possibly counter-intuitive properties of RFC-5545-compliant rrules, and other documentation improvements in the rrule module (gh issue #105, gh issue #149 - pointer to the solution by @phep, pr #213).

10.1.14 Version 2.5.0

- Updated zoneinfo to 2016a
- zoneinfo_metadata file version increased to 2.0 - the updated updatezinfo.py script will work with older zoneinfo_metadata.json files, but new metadata files will not work with older updatezinfo.py versions. Additionally, we have started hosting our own mirror of the Olson databases on a GitHub pages site (https://dateutil.github.io/tzdata/) (gh pr #183)
- dateutil zoneinfo tarballs now contain the full zoneinfo_metadata file used to generate them. (gh issue #27, gh pr #85)
- relativedelta can now be safely subclassed without derived objects reverting to base relativedelta objects as a result of arithmetic operations. (lp:1010199, gh issue #44, pr #49)
- relativedelta 'weeks' parameter can now be set and retrieved as a property of relativedelta instances. (lp: 727525, gh issue #45, pr #49)
- relativedelta now explicitly supports fractional relative weeks, days, hours, minutes and seconds. Fractional values in absolute parameters (year, day, etc) are now deprecated. (gh issue #40, pr #190)
- relativedelta objects previously did not use microseconds to determine of two relativedelta objects were equal. This oversight has been corrected. Contributed by @elprans (gh pr #113)

- rrule now has an xafter() method for retrieving multiple recurrences after a specified date. (gh pr #38)
- str(rrule) now returns an RFC2445-compliant rrule string, contributed by @schinckel and @armicron (lp:1406305, gh issue #47, prs #50, #62 and #160)
- rrule performance under certain conditions has been significantly improved thanks to a patch contributed by @dekoza, based on an article by Brian Beck (@exogen) (gh pr #136)
- The use of both the 'until' and 'count' parameters is now deprecated as inconsistent with RFC2445 (gh pr #62, #185)
- Parsing an empty string will now raise a ValueError, rather than returning the datetime passed to the 'default' parameter. (gh issue #78, pr #187)
- tzwinlocal objects now have a meaningful repr() and str() implementation (gh issue #148, prs #184 and #186)
- Added equality logic for tzwin and tzwinlocal objects. (gh issue #151, pr #180, #184)
- Added some flexibility in subclassing timelex, and switched the default behavior over to using string methods rather than comparing against a fixed list. (gh pr #122, #139)
- An issue causing tzstr() to crash on Python 2.x was fixed. (lp: 1331576, gh issue #51, pr #55)
- An issue with string encoding causing exceptions under certain circumstances when tzname() is called was fixed. (gh issue #60, #74, pr #75)
- Parser issue where calling parse() on dates with no day specified when the day of the month in the default datetime (which is "today" if unspecified) is greater than the number of days in the parsed month was fixed (this issue tended to crop up between the 29th and 31st of the month, for obvious reasons) (canonical gh issue #25, pr #30, #191)
- Fixed parser issue causing fuzzy_with_tokens to raise an unexpected exception in certain circumstances. Contributed by @MichaelAquilina (gh pr #91)
- Fixed parser issue where years > 100 AD were incorrectly parsed. Contributed by @Bachmann1234 (gh pr #130)
- Fixed parser issue where commas were not a valid separator between seconds and microseconds, preventing parsing of ISO 8601 dates. Contributed by @ryanss (gh issue #28, pr #106)
- Fixed issue with tzwin encoding in locales with non-Latin alphabets (gh issue #92, pr #98)
- Fixed an issue where tzwin was not being properly imported on Windows. Contributed by @labrys. (gh pr #134)
- Fixed a problem causing issues importing zoneinfo in certain circumstances. Issue and solution contributed by @alexxv (gh issue #97, pr #99)
- Fixed an issue where dateutil timezones were not compatible with basic time objects. One of many, many timezone related issues contributed and tested by @labrys. (gh issue #132, pr #181)
- Fixed issue where tzwinlocal had an invalid utcoffset. (gh issue #135, pr #141, #142)
- Fixed issue with tzwin and tzwinlocal where DST transitions were incorrectly parsed from the registry. (gh issue #143, pr #178)
- updatezinfo.py no longer suppresses certain OSErrors. Contributed by @bjamesv (gh pr #164)
- An issue that arose when timezone locale changes during runtime has been fixed by @carlosxl and @mjschultz (gh issue #100, prs #107, #109)
- Python 3.5 was added to the supported platforms in the metadata (@tacaswell gh pr #159) and the test suites (@moreati gh pr #117).
- An issue with tox failing without unittest2 installed in Python 2.6 was fixed by @moreati (gh pr #115)

- Several deprecated functions were replaced in the tests by @moreati (gh pr #116)
- Improved the logic in Travis and Appveyor to alleviate issues where builds were failing due to connection issues when downloading the IANA timezone files. In addition to adding our own mirror for the files (gh pr #183), the download is now retried a number of times (with a delay) (gh pr #177)
- Many failing doctests were fixed by @moreati. (gh pr #120)
- Many fixes to the documentation (gh pr #103, gh pr #87 from @radarhere, gh pr #154 from @gpoesia, gh pr #156 from @awsum, gh pr #168 from @ja8zyjits)
- Added a code coverage tool to the CI to help improve the library. (gh pr #182)
- We now have a mailing list - dateutil@python.org, graciously hosted by Python.org.

10.1.15 Version 2.4.2

- Updated zoneinfo to 2015b.
- Fixed issue with parsing of tzstr on Python 2.7.x; tzstr will now be decoded if not a unicode type. gh #51 (lp:1331576), gh pr #55.
- Fix a parser issue where AM and PM tokens were showing up in fuzzy date stamps, triggering inappropriate errors. gh #56 (lp: 1428895), gh pr #63.
- Missing function "setcachesize" removed from zoneinfo __all__ list by @ryanss, fixing an issue with wildcard imports of dateutil.zoneinfo. (gh pr #66).
- (PyPI only) Fix an issue with source distributions not including the test suite.

10.1.16 Version 2.4.1

- Added explicit check for valid hours if AM/PM is specified in parser. (gh pr #22, issue #21)
- Fix bug in rrule introduced in 2.4.0 where byweekday parameter was not handled properly. (gh pr #35, issue #34)
- Fix error where parser allowed some invalid dates, overwriting existing hours with the last 2-digit number in the string. (gh pr #32, issue #31)
- Fix and add test for Python 2.x compatibility with boolean checking of relativedelta objects. Implemented by @nimasmi (gh pr #43) and Cédric Krier (lp: 1035038)
- Replaced parse() calls with explicit datetime objects in unit tests unrelated to parser. (gh pr #36)
- Changed private _byxxx from sets to sorted tuples and fixed one currently unreachable bug in _construct_byset. (gh pr #54)
- Additional documentation for parser (gh pr #29, #33, #41) and rrule.
- Formatting fixes to documentation of rrule and README.rst.
- Updated zoneinfo to 2015a.

10.1.17 Version 2.4.0

- Fix an issue with relativedelta and freezegun (lp:1374022)
- Fix tzinfo in windows for timezones without dst (lp:1010050, gh #2)
- Ignore missing timezones in windows like in POSIX

- Fix minimal version requirement for six (gh #6)
- **Many rrule changes and fixes by @pganssle (gh pull requests #13 #14 #17),** including defusing some infinite loops (gh #4)

10.1.18 Version 2.3

- Cleanup directory structure, moved test.py to dateutil/tests/test.py
- Changed many aspects of dealing with the zone info file. Instead of a cache, all the zones are loaded to memory, but symbolic links are loaded only once, so not much memory is used.
- The package is now zip-safe, and universal-wheelable, thanks to changes in the handling of the zoneinfo file.
- Fixed tzwin silently not imported on windows python2
- New maintainer, together with new hosting: GitHub, Travis, Read-The-Docs

10.1.19 Version 2.2

- Updated zoneinfo to 2013h
- fuzzy_with_tokens parse addon from Christopher Corley
- Bug with LANG=C fixed by Mike Gilbert

10.1.20 Version 2.1

- New maintainer
- Dateutil now works on Python 2.6, 2.7 and 3.2 from same codebase (with six)
- #704047: Ismael Carnales' patch for a new time format
- Small bug fixes, thanks for reporters!

10.1.21 Version 2.0

- Ported to Python 3, by Brian Jones. If you need dateutil for Python 2.X, please continue using the 1.X series.
- There's no such thing as a "PSF License". This source code is now made available under the Simplified BSD license. See LICENSE for details.

10.1.22 Version 1.5

- As reported by Mathieu Bridon, rrules were matching the bysecond rules incorrectly against byminute in some circumstances when the SECONDLY frequency was in use, due to a copy & paste bug. The problem has been unittested and corrected.
- Adam Ryan reported a problem in the relativedelta implementation which affected the yearday parameter in the month of January specifically. This has been unittested and fixed.
- Updated timezone information.

10.1.23 Version 1.4.1

- Updated timezone information.

10.1.24 Version 1.4

- Fixed another parser precision problem on conversion of decimal seconds to microseconds, as reported by Erik Brown. Now these issues are gone for real since it's not using floating point arithmetic anymore.
- Fixed case where tzrange.utcoffset and tzrange.dst() might fail due to a date being used where a datetime was expected (reported and fixed by Lennart Regebro).
- Prevent tzstr from introducing daylight timings in strings that didn't specify them (reported by Lennart Regebro).
- Calls like gettz("GMT+3") and gettz("UTC-2") will now return the expected values, instead of the TZ variable behavior.
- Fixed DST signal handling in zoneinfo files. Reported by Nicholas F. Fabry and John-Mark Gurney.

10.1.25 Version 1.3

- Fixed precision problem on conversion of decimal seconds to microseconds, as reported by Skip Montanaro.
- Fixed bug in constructor of parser, and converted parser classes to new-style classes. Original report and patch by Michael Elsdörfer.
- Initialize tzid and comps in tz.py, to prevent the code from ever raising a NameError (even with broken files). Johan Dahlin suggested the fix after a pyflakes run.
- Version is now published in dateutil.__version__, as requested by Darren Dale.
- All code is compatible with new-style division.

10.1.26 Version 1.2

- Now tzfile will round timezones to full-minutes if necessary, since Python's datetime doesn't support sub-minute offsets. Thanks to Ilpo Nyyssönen for reporting the issue.
- Removed bare string exceptions, as reported and fixed by Wilfredo Sánchez Vega.
- Fix bug in leap count parsing (reported and fixed by Eugene Oden).

10.1.27 Version 1.1

- Fixed rrule byyearday handling. Abramo Bagnara pointed out that RFC2445 allows negative numbers.
- Fixed –prefix handling in setup.py (by Sidnei da Silva).
- Now tz.gettz() returns a tzlocal instance when not given any arguments and no other timezone information is found.
- Updating timezone information to version 2005q.

10.1.28 Version 1.0

- Fixed parsing of XXhXXm formatted time after day/month/year has been parsed.
- Added patch by Jeffrey Harris optimizing rrule.__contains__.

10.1.29 Version 0.9

- Fixed pickling of timezone types, as reported by Andreas Köhler.
- Implemented internal timezone information with binary timezone files. datautil.tz.gettz() function will now try to use the system timezone files, and fallback to the internal versions. It's also possible to ask for the internal versions directly by using dateutil.zoneinfo.gettz().
- New tzwin timezone type, allowing access to Windows internal timezones (contributed by Jeffrey Harris).
- Fixed parsing of unicode date strings.
- Accept parserinfo instances as the parser constructor parameter, besides parserinfo (sub)classes.
- Changed weekday to spell the not-set n value as None instead of 0.
- Fixed other reported bugs.

10.1.30 Version 0.5

- Removed `FREQ_` prefix from rrule frequency constants WARNING: this breaks compatibility with previous versions.
- Fixed rrule.between() for cases where "after" is achieved before even starting, as reported by Andreas Köhler.
- Fixed two digit zero-year parsing (such as 31-Dec-00), as reported by Jim Abramson, and included test case for this.
- Sort exdate and rdate before iterating over them, so that it's not necessary to sort them before adding to the rruleset, as reported by Nicholas Piper.

10.2 dateutil examples

Contents

10.2.1 relativedelta examples

Let's begin our trip:

```
>>> from datetime import *; from dateutil.relativedelta import *
>>> import calendar
```

Store some values:

```
>>> NOW = datetime.now()
>>> TODAY = date.today()
>>> NOW
datetime.datetime(2003, 9, 17, 20, 54, 47, 282310)
>>> TODAY
datetime.date(2003, 9, 17)
```

Next month

```
>>> NOW+relativedelta(months=+1)
datetime.datetime(2003, 10, 17, 20, 54, 47, 282310)
```

Next month, plus one week.

```
>>> NOW+relativedelta(months=+1, weeks=+1)
datetime.datetime(2003, 10, 24, 20, 54, 47, 282310)
```

Next month, plus one week, at 10am.

```
>>> TODAY+relativedelta(months=+1, weeks=+1, hour=10)
datetime.datetime(2003, 10, 24, 10, 0)
```

Here is another example using an absolute relativedelta. Notice the use of year and month (both singular) which causes the values to be *replaced* in the original datetime rather than performing an arithmetic operation on them.

```
>>> NOW+relativedelta(year=1, month=1)
datetime.datetime(1, 1, 17, 20, 54, 47, 282310)
```

Let's try the other way around. Notice that the hour setting we get in the relativedelta is relative, since it's a difference, and the weeks parameter has gone.

```
>>> relativedelta(datetime(2003, 10, 24, 10, 0), TODAY)
relativedelta(months=+1, days=+7, hours=+10)
```

One month before one year.

```
>>> NOW+relativedelta(years=+1, months=-1)
datetime.datetime(2004, 8, 17, 20, 54, 47, 282310)
```

How does it handle months with different numbers of days? Notice that adding one month will never cross the month boundary.

```
>>> date(2003,1,27)+relativedelta(months=+1)
datetime.date(2003, 2, 27)
>>> date(2003,1,31)+relativedelta(months=+1)
datetime.date(2003, 2, 28)
>>> date(2003,1,31)+relativedelta(months=+2)
datetime.date(2003, 3, 31)
```

The logic for years is the same, even on leap years.

```
>>> date(2000,2,28)+relativedelta(years=+1)
datetime.date(2001, 2, 28)
>>> date(2000,2,29)+relativedelta(years=+1)
datetime.date(2001, 2, 28)

>>> date(1999,2,28)+relativedelta(years=+1)
datetime.date(2000, 2, 28)
>>> date(1999,3,1)+relativedelta(years=+1)
datetime.date(2000, 3, 1)

>>> date(2001,2,28)+relativedelta(years=-1)
datetime.date(2000, 2, 28)
>>> date(2001,3,1)+relativedelta(years=-1)
datetime.date(2000, 3, 1)
```

Next friday

```
>>> TODAY+relativedelta(weekday=FR)
datetime.date(2003, 9, 19)

>>> TODAY+relativedelta(weekday=calendar.FRIDAY)
datetime.date(2003, 9, 19)
```

Last friday in this month.

```
>>> TODAY+relativedelta(day=31, weekday=FR(-1))
datetime.date(2003, 9, 26)
```

Next wednesday (it's today!).

```
>>> TODAY+relativedelta(weekday=WE(+1))
datetime.date(2003, 9, 17)
```

Next wednesday, but not today.

```
>>> TODAY+relativedelta(days=+1, weekday=WE(+1))
datetime.date(2003, 9, 24)
```

Following ISO year week number notation find the first day of the 15th week of 1997.

```
>>> datetime(1997,1,1)+relativedelta(day=4, weekday=MO(-1), weeks=+14)
datetime.datetime(1997, 4, 7, 0, 0)
```

How long ago has the millennium changed?

```
>>> relativedelta(NOW, date(2001,1,1))
relativedelta(years=+2, months=+8, days=+16,
              hours=+20, minutes=+54, seconds=+47, microseconds=+282310)
```

How old is John?

```
>>> johnbirthday = datetime(1978, 4, 5, 12, 0)
>>> relativedelta(NOW, johnbirthday)
relativedelta(years=+25, months=+5, days=+12,
          hours=+8, minutes=+54, seconds=+47, microseconds=+282310)
```

It works with dates too.

```
>>> relativedelta(TODAY, johnbirthday)
relativedelta(years=+25, months=+5, days=+11, hours=+12)
```

Obtain today's date using the yearday:

```
>>> date(2003, 1, 1)+relativedelta(yearday=260)
datetime.date(2003, 9, 17)
```

We can use today's date, since yearday should be absolute in the given year:

```
>>> TODAY+relativedelta(yearday=260)
datetime.date(2003, 9, 17)
```

Last year it should be in the same day:

```
>>> date(2002, 1, 1)+relativedelta(yearday=260)
datetime.date(2002, 9, 17)
```

But not in a leap year:

```
>>> date(2000, 1, 1)+relativedelta(yearday=260)
datetime.date(2000, 9, 16)
```

We can use the non-leap year day to ignore this:

```
>>> date(2000, 1, 1)+relativedelta(nlyearday=260)
datetime.date(2000, 9, 17)
```

10.2.2 rrule examples

These examples were converted from the RFC.

Prepare the environment.

```
>>> from dateutil.rrule import *
>>> from dateutil.parser import *
>>> from datetime import *

>>> import pprint
>>> import sys
>>> sys.displayhook = pprint.pprint
```

Daily, for 10 occurrences.

```
>>> list(rrule(DAILY, count=10,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 9, 6, 9, 0),
 datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 10, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0)]
```

Daily until December 24, 1997

```
>>> list(rrule(DAILY,
...            dtstart=parse("19970902T090000"),
...            until=parse("19971224T000000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 ...
 datetime.datetime(1997, 12, 21, 9, 0),
 datetime.datetime(1997, 12, 22, 9, 0),
 datetime.datetime(1997, 12, 23, 9, 0)]
```

Every other day, 5 occurrences.

```
>>> list(rrule(DAILY, interval=2, count=5,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 6, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0),
 datetime.datetime(1997, 9, 10, 9, 0)]
```

Every 10 days, 5 occurrences.

```
>>> list(rrule(DAILY, interval=10, count=5,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Everyday in January, for 3 years.

```
>>> list(rrule(YEARLY, bymonth=1, byweekday=range(7),
...            dtstart=parse("19980101T090000"),
...            until=parse("20000131T090000")))
[datetime.datetime(1998, 1, 1, 9, 0),
 datetime.datetime(1998, 1, 2, 9, 0),
 ...
 datetime.datetime(1998, 1, 30, 9, 0),
 datetime.datetime(1998, 1, 31, 9, 0),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1999, 1, 1, 9, 0),
 datetime.datetime(1999, 1, 2, 9, 0),
 ...
 datetime.datetime(1999, 1, 30, 9, 0),
 datetime.datetime(1999, 1, 31, 9, 0),
 datetime.datetime(2000, 1, 1, 9, 0),
 datetime.datetime(2000, 1, 2, 9, 0),
 ...
 datetime.datetime(2000, 1, 30, 9, 0),
 datetime.datetime(2000, 1, 31, 9, 0)]
```

Same thing, in another way.

```
>>> list(rrule(DAILY, bymonth=1,
...            dtstart=parse("19980101T090000"),
...            until=parse("20000131T090000")))
[datetime.datetime(1998, 1, 1, 9, 0),
 ...
 datetime.datetime(2000, 1, 31, 9, 0)]
```

Weekly for 10 occurrences.

```
>>> list(rrule(WEEKLY, count=10,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 7, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 21, 9, 0),
 datetime.datetime(1997, 10, 28, 9, 0),
 datetime.datetime(1997, 11, 4, 9, 0)]
```

Every other week, 6 occurrences.

```
>>> list(rrule(WEEKLY, interval=2, count=6,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 28, 9, 0),
 datetime.datetime(1997, 11, 11, 9, 0)]
```

Weekly on Tuesday and Thursday for 5 weeks.

```
>>> list(rrule(WEEKLY, count=10, wkst=SU, byweekday=(TU,TH),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 18, 9, 0),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 25, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0)]
```

Every other week on Tuesday and Thursday, for 8 occurrences.

```
>>> list(rrule(WEEKLY, interval=2, count=8,
...            wkst=SU, byweekday=(TU,TH),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 18, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 16, 9, 0)]
```

Monthly on the 1st Friday for ten occurrences.

```
>>> list(rrule(MONTHLY, count=10, byweekday=FR(1),
...            dtstart=parse("19970905T090000")))
[datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 10, 3, 9, 0),
 datetime.datetime(1997, 11, 7, 9, 0),
 datetime.datetime(1997, 12, 5, 9, 0),
 datetime.datetime(1998, 1, 2, 9, 0),
 datetime.datetime(1998, 2, 6, 9, 0),
 datetime.datetime(1998, 3, 6, 9, 0),
 datetime.datetime(1998, 4, 3, 9, 0),
 datetime.datetime(1998, 5, 1, 9, 0),
 datetime.datetime(1998, 6, 5, 9, 0)]
```

Every other month on the 1st and last Sunday of the month for 10 occurrences.

```
>>> list(rrule(MONTHLY, interval=2, count=10,
...            byweekday=(SU(1), SU(-1)),
...            dtstart=parse("19970907T090000")))
[datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 28, 9, 0),
 datetime.datetime(1997, 11, 2, 9, 0),
 datetime.datetime(1997, 11, 30, 9, 0),
 datetime.datetime(1998, 1, 4, 9, 0),
 datetime.datetime(1998, 1, 25, 9, 0),
 datetime.datetime(1998, 3, 1, 9, 0),
 datetime.datetime(1998, 3, 29, 9, 0),
 datetime.datetime(1998, 5, 3, 9, 0),
 datetime.datetime(1998, 5, 31, 9, 0)]
```

Monthly on the second to last Monday of the month for 6 months.

```
>>> list(rrule(MONTHLY, count=6, byweekday=MO(-2),
...            dtstart=parse("19970922T090000")))
[datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 20, 9, 0),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1997, 11, 17, 9, 0),
 datetime.datetime(1997, 12, 22, 9, 0),
 datetime.datetime(1998, 1, 19, 9, 0),
 datetime.datetime(1998, 2, 16, 9, 0)]
```

Monthly on the third to the last day of the month, for 6 months.

```
>>> list(rrule(MONTHLY, count=6, bymonthday=-3,
...            dtstart=parse("19970928T090000")))
[datetime.datetime(1997, 9, 28, 9, 0),
 datetime.datetime(1997, 10, 29, 9, 0),
 datetime.datetime(1997, 11, 28, 9, 0),
 datetime.datetime(1997, 12, 29, 9, 0),
 datetime.datetime(1998, 1, 29, 9, 0),
 datetime.datetime(1998, 2, 26, 9, 0)]
```

Monthly on the 2nd and 15th of the month for 5 occurrences.

```
>>> list(rrule(MONTHLY, count=5, bymonthday=(2,15),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 15, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 15, 9, 0),
 datetime.datetime(1997, 11, 2, 9, 0)]
```

Monthly on the first and last day of the month for 3 occurrences.

```
>>> list(rrule(MONTHLY, count=5, bymonthday=(-1,1,),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 1, 9, 0),
 datetime.datetime(1997, 10, 31, 9, 0),
 datetime.datetime(1997, 11, 1, 9, 0),
 datetime.datetime(1997, 11, 30, 9, 0)]
```

Every 18 months on the 10th thru 15th of the month for 10 occurrences.

```
>>> list(rrule(MONTHLY, interval=18, count=10,
...            bymonthday=range(10,16),
...            dtstart=parse("19970910T090000")))
[datetime.datetime(1997, 9, 10, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 13, 9, 0),
 datetime.datetime(1997, 9, 14, 9, 0),
 datetime.datetime(1997, 9, 15, 9, 0),
 datetime.datetime(1999, 3, 10, 9, 0),
 datetime.datetime(1999, 3, 11, 9, 0),
 datetime.datetime(1999, 3, 12, 9, 0),
 datetime.datetime(1999, 3, 13, 9, 0)]
```

Every Tuesday, every other month, 6 occurrences.

```
>>> list(rrule(MONTHLY, interval=2, count=6, byweekday=TU,
...            dtstart=parse("19970902T090000")))
```

(continues on next page)

(continued from previous page)

```
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 11, 4, 9, 0)]
```

Yearly in June and July for 10 occurrences.

```
>>> list(rrule(YEARLY, count=4, bymonth=(6,7),
...            dtstart=parse("19970610T090000")))
[datetime.datetime(1997, 6, 10, 9, 0),
 datetime.datetime(1997, 7, 10, 9, 0),
 datetime.datetime(1998, 6, 10, 9, 0),
 datetime.datetime(1998, 7, 10, 9, 0)]
```

Every 3rd year on the 1st, 100th and 200th day for 4 occurrences.

```
>>> list(rrule(YEARLY, count=4, interval=3, byyearday=(1,100,200),
...            dtstart=parse("19970101T090000")))
[datetime.datetime(1997, 1, 1, 9, 0),
 datetime.datetime(1997, 4, 10, 9, 0),
 datetime.datetime(1997, 7, 19, 9, 0),
 datetime.datetime(2000, 1, 1, 9, 0)]
```

Every 20th Monday of the year, 3 occurrences.

```
>>> list(rrule(YEARLY, count=3, byweekday=MO(20),
...            dtstart=parse("19970519T090000")))
[datetime.datetime(1997, 5, 19, 9, 0),
 datetime.datetime(1998, 5, 18, 9, 0),
 datetime.datetime(1999, 5, 17, 9, 0)]
```

Monday of week number 20 (where the default start of the week is Monday), 3 occurrences.

```
>>> list(rrule(YEARLY, count=3, byweekno=20, byweekday=MO,
...            dtstart=parse("19970512T090000")))
[datetime.datetime(1997, 5, 12, 9, 0),
 datetime.datetime(1998, 5, 11, 9, 0),
 datetime.datetime(1999, 5, 17, 9, 0)]
```

The week number 1 may be in the last year.

```
>>> list(rrule(WEEKLY, count=3, byweekno=1, byweekday=MO,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 12, 29, 9, 0),
 datetime.datetime(1999, 1, 4, 9, 0),
 datetime.datetime(2000, 1, 3, 9, 0)]
```

And the week numbers greater than 51 may be in the next year.

```
>>> list(rrule(WEEKLY, count=3, byweekno=52, byweekday=SU,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 12, 28, 9, 0),
 datetime.datetime(1998, 12, 27, 9, 0),
 datetime.datetime(2000, 1, 2, 9, 0)]
```

Only some years have week number 53:

```
>>> list(rrule(WEEKLY, count=3, byweekno=53, byweekday=MO,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1998, 12, 28, 9, 0),
 datetime.datetime(2004, 12, 27, 9, 0),
 datetime.datetime(2009, 12, 28, 9, 0)]
```

Every Friday the 13th, 4 occurrences.

```
>>> list(rrule(YEARLY, count=4, byweekday=FR, bymonthday=13,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1998, 2, 13, 9, 0),
 datetime.datetime(1998, 3, 13, 9, 0),
 datetime.datetime(1998, 11, 13, 9, 0),
 datetime.datetime(1999, 8, 13, 9, 0)]
```

Every four years, the first Tuesday after a Monday in November, 3 occurrences (U.S. Presidential Election day):

```
>>> list(rrule(YEARLY, interval=4, count=3, bymonth=11,
...            byweekday=TU, bymonthday=(2,3,4,5,6,7,8),
...            dtstart=parse("19961105T090000")))
[datetime.datetime(1996, 11, 5, 9, 0),
 datetime.datetime(2000, 11, 7, 9, 0),
 datetime.datetime(2004, 11, 2, 9, 0)]
```

The 3rd instance into the month of one of Tuesday, Wednesday or Thursday, for the next 3 months:

```
>>> list(rrule(MONTHLY, count=3, byweekday=(TU,WE,TH),
...            bysetpos=3, dtstart=parse("19970904T090000")))
[datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 10, 7, 9, 0),
 datetime.datetime(1997, 11, 6, 9, 0)]
```

The 2nd to last weekday of the month, 3 occurrences.

```
>>> list(rrule(MONTHLY, count=3, byweekday=(MO,TU,WE,TH,FR),
...            bysetpos=-2, dtstart=parse("19970929T090000")))
[datetime.datetime(1997, 9, 29, 9, 0),
 datetime.datetime(1997, 10, 30, 9, 0),
 datetime.datetime(1997, 11, 27, 9, 0)]
```

Every 3 hours from 9:00 AM to 5:00 PM on a specific day.

```
>>> list(rrule(HOURLY, interval=3,
...            dtstart=parse("19970902T090000"),
...            until=parse("19970902T170000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 12, 0),
 datetime.datetime(1997, 9, 2, 15, 0)]
```

Every 15 minutes for 6 occurrences.

```
>>> list(rrule(MINUTELY, interval=15, count=6,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 9, 15),
 datetime.datetime(1997, 9, 2, 9, 30),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1997, 9, 2, 9, 45),
 datetime.datetime(1997, 9, 2, 10, 0),
 datetime.datetime(1997, 9, 2, 10, 15)]
```

Every hour and a half for 4 occurrences.

```
>>> list(rrule(MINUTELY, interval=90, count=4,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 10, 30),
 datetime.datetime(1997, 9, 2, 12, 0),
 datetime.datetime(1997, 9, 2, 13, 30)]
```

Every 20 minutes from 9:00 AM to 4:40 PM for two days.

```
>>> list(rrule(MINUTELY, interval=20, count=48,
...            byhour=range(9,17), byminute=(0,20,40),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 9, 20),
 ...
 datetime.datetime(1997, 9, 2, 16, 20),
 datetime.datetime(1997, 9, 2, 16, 40),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 20),
 ...
 datetime.datetime(1997, 9, 3, 16, 20),
 datetime.datetime(1997, 9, 3, 16, 40)]
```

An example where the days generated makes a difference because of *wkst*.

```
>>> list(rrule(WEEKLY, interval=2, count=4,
...            byweekday=(TU,SU), wkst=MO,
...            dtstart=parse("19970805T090000")))
[datetime.datetime(1997, 8, 5, 9, 0),
 datetime.datetime(1997, 8, 10, 9, 0),
 datetime.datetime(1997, 8, 19, 9, 0),
 datetime.datetime(1997, 8, 24, 9, 0)]

>>> list(rrule(WEEKLY, interval=2, count=4,
...            byweekday=(TU,SU), wkst=SU,
...            dtstart=parse("19970805T090000")))
[datetime.datetime(1997, 8, 5, 9, 0),
 datetime.datetime(1997, 8, 17, 9, 0),
 datetime.datetime(1997, 8, 19, 9, 0),
 datetime.datetime(1997, 8, 31, 9, 0)]
```

10.2.3 rruleset examples

Daily, for 7 days, jumping Saturday and Sunday occurrences.

```
>>> set = rruleset()
>>> set.rrule(rrule(DAILY, count=7,
...                 dtstart=parse("19970902T090000")))
>>> set.exrule(rrule(YEARLY, byweekday=(SA,SU),
```

(continues on next page)

(continued from previous page)

```
...                   dtstart=parse("19970902T090000")))
>>> list(set)
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0)]
```

Weekly, for 4 weeks, plus one time on day 7, and not on day 16.

```
>>> set = rruleset()
>>> set.rrule(rrule(WEEKLY, count=4,
...           dtstart=parse("19970902T090000")))
>>> set.rdate(datetime.datetime(1997, 9, 7, 9, 0))
>>> set.exdate(datetime.datetime(1997, 9, 16, 9, 0))
>>> list(set)
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0)]
```

10.2.4 rrulestr() examples

Every 10 days, 5 occurrences.

```
>>> list(rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... """))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Same thing, but passing only the *RRULE* value.

```
>>> list(rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5",
...               dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Notice that when using a single rule, it returns an *rrule* instance, unless *forceset* was used.

```
>>> rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5")
<dateutil.rrule.rrule object at 0x...>

>>> rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... """)
<dateutil.rrule.rrule object at 0x...>
```

(continues on next page)

(continued from previous page)

```
>>> rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5", forceset=True)
<dateutil.rrule.rruleset object at 0x...>
```

But when an *rruleset* is needed, it is automatically used.

```
>>> rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... RRULE:FREQ=DAILY;INTERVAL=5;COUNT=3
... """)
<dateutil.rrule.rruleset object at 0x...>
```

10.2.5 parse examples

The following code will prepare the environment:

```
>>> from dateutil.parser import *
>>> from dateutil.tz import *
>>> from datetime import *
>>> TZOFFSETS = {"BRST": -10800}
>>> BRSTTZ = tzoffset("BRST", -10800)
>>> DEFAULT = datetime(2003, 9, 25)
```

Some simple examples based on the *date* command, using the *ZOFFSET* dictionary to provide the BRST timezone offset.

```
>>> parse("Thu Sep 25 10:36:28 BRST 2003", tzinfos=TZOFFSETS)
datetime.datetime(2003, 9, 25, 10, 36, 28,
          tzinfo=tzoffset('BRST', -10800))

>>> parse("2003 10:36:28 BRST 25 Sep Thu", tzinfos=TZOFFSETS)
datetime.datetime(2003, 9, 25, 10, 36, 28,
          tzinfo=tzoffset('BRST', -10800))
```

Notice that since BRST is my local timezone, parsing it without further timezone settings will yield a *tzlocal* timezone.

```
>>> parse("Thu Sep 25 10:36:28 BRST 2003")
datetime.datetime(2003, 9, 25, 10, 36, 28, tzinfo=tzlocal())
```

We can also ask to ignore the timezone explicitly:

```
>>> parse("Thu Sep 25 10:36:28 BRST 2003", ignoretz=True)
datetime.datetime(2003, 9, 25, 10, 36, 28)
```

That's the same as processing a string without timezone:

```
>>> parse("Thu Sep 25 10:36:28 2003")
datetime.datetime(2003, 9, 25, 10, 36, 28)
```

Without the year, but passing our *DEFAULT* datetime to return the same year, no mattering what year we currently are in:

```
>>> parse("Thu Sep 25 10:36:28", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28)
```

Strip it further:

```
>>> parse("Thu Sep 10:36:28", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28)

>>> parse("Thu 10:36:28", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28)

>>> parse("Thu 10:36", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36)

>>> parse("10:36", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36)
```

Strip in a different way:

```
>>> parse("Thu Sep 25 2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep 25 2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep 2003", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)
```

Another format, based on *date -R* (RFC822):

```
>>> parse("Thu, 25 Sep 2003 10:49:41 -0300")
datetime.datetime(2003, 9, 25, 10, 49, 41,
          tzinfo=tzoffset(None, -10800))
```

ISO format:

```
>>> parse("2003-09-25T10:49:41.5-03:00")
datetime.datetime(2003, 9, 25, 10, 49, 41, 500000,
          tzinfo=tzoffset(None, -10800))
```

Some variations:

```
>>> parse("2003-09-25T10:49:41")
datetime.datetime(2003, 9, 25, 10, 49, 41)

>>> parse("2003-09-25T10:49")
datetime.datetime(2003, 9, 25, 10, 49)

>>> parse("2003-09-25T10")
datetime.datetime(2003, 9, 25, 10, 0)

>>> parse("2003-09-25")
datetime.datetime(2003, 9, 25, 0, 0)
```

ISO format, without separators:

```
>>> parse("20030925T104941.5-0300")
datetime.datetime(2003, 9, 25, 10, 49, 41, 500000,
                  tzinfo=tzoffset(None, -10800))

>>> parse("20030925T104941-0300")
datetime.datetime(2003, 9, 25, 10, 49, 41,
          tzinfo=tzoffset(None, -10800))

>>> parse("20030925T104941")
datetime.datetime(2003, 9, 25, 10, 49, 41)

>>> parse("20030925T1049")
datetime.datetime(2003, 9, 25, 10, 49)

>>> parse("20030925T10")
datetime.datetime(2003, 9, 25, 10, 0)

>>> parse("20030925")
datetime.datetime(2003, 9, 25, 0, 0)
```

Everything together.

```
>>> parse("199709020900")
datetime.datetime(1997, 9, 2, 9, 0)
>>> parse("19970902090059")
datetime.datetime(1997, 9, 2, 9, 0, 59)
```

Different date orderings:

```
>>> parse("2003-09-25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003-Sep-25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("25-Sep-2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("Sep-25-2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("09-25-2003")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("25-09-2003")
datetime.datetime(2003, 9, 25, 0, 0)
```

Check some ambiguous dates:

```
>>> parse("10-09-2003")
datetime.datetime(2003, 10, 9, 0, 0)

>>> parse("10-09-2003", dayfirst=True)
datetime.datetime(2003, 9, 10, 0, 0)

>>> parse("10-09-03")
datetime.datetime(2003, 10, 9, 0, 0)
```

(continues on next page)

(continued from previous page)

```
>>> parse("10-09-03", yearfirst=True)
datetime.datetime(2010, 9, 3, 0, 0)
```

Other date separators are allowed:

```
>>> parse("2003.Sep.25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003/09/25")
datetime.datetime(2003, 9, 25, 0, 0)
```

Even with spaces:

```
>>> parse("2003 Sep 25")
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("2003 09 25")
datetime.datetime(2003, 9, 25, 0, 0)
```

Hours with letters work:

```
>>> parse("10h36m28.5s", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 36, 28, 500000)

>>> parse("01s02h03m", default=DEFAULT)
datetime.datetime(2003, 9, 25, 2, 3, 1)

>>> parse("01h02m03", default=DEFAULT)
datetime.datetime(2003, 9, 25, 1, 2, 3)

>>> parse("01h02", default=DEFAULT)
datetime.datetime(2003, 9, 25, 1, 2)

>>> parse("01h02s", default=DEFAULT)
datetime.datetime(2003, 9, 25, 1, 0, 2)
```

With AM/PM:

```
>>> parse("10h am", default=DEFAULT)
datetime.datetime(2003, 9, 25, 10, 0)

>>> parse("10pm", default=DEFAULT)
datetime.datetime(2003, 9, 25, 22, 0)

>>> parse("12:00am", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)

>>> parse("12pm", default=DEFAULT)
datetime.datetime(2003, 9, 25, 12, 0)
```

Some special treating for ‘’pertain” relations:

```
>>> parse("Sep 03", default=DEFAULT)
datetime.datetime(2003, 9, 3, 0, 0)
```

(continues on next page)

(continued from previous page)

```
>>> parse("Sep of 03", default=DEFAULT)
datetime.datetime(2003, 9, 25, 0, 0)
```

Fuzzy parsing:

```
>>> s = "Today is 25 of September of 2003, exactly " \
...     "at 10:49:41 with timezone -03:00."
>>> parse(s, fuzzy=True)
datetime.datetime(2003, 9, 25, 10, 49, 41,
          tzinfo=tzoffset(None, -10800))
```

Other random formats:

```
>>> parse("Wed, July 10, '96")
datetime.datetime(1996, 7, 10, 0, 0)

>>> parse("1996.07.10 AD at 15:08:56 PDT", ignoretz=True)
datetime.datetime(1996, 7, 10, 15, 8, 56)

>>> parse("Tuesday, April 12, 1952 AD 3:30:42pm PST", ignoretz=True)
datetime.datetime(1952, 4, 12, 15, 30, 42)

>>> parse("November 5, 1994, 8:15:30 am EST", ignoretz=True)
datetime.datetime(1994, 11, 5, 8, 15, 30)

>>> parse("3rd of May 2001")
datetime.datetime(2001, 5, 3, 0, 0)

>>> parse("5:50 A.M. on June 13, 1990")
datetime.datetime(1990, 6, 13, 5, 50)
```

Override parserinfo with a custom parserinfo

```
>>> from dateutil.parser import parse, parserinfo
>>> class CustomParserInfo(parserinfo):
...     # e.g. edit a property of parserinfo to allow a custom 12 hour format
...     AMPM = [("am", "a", "xm"), ("pm", "p")]
>>> parse('2018-06-08 12:06:58 XM', parserinfo=CustomParserInfo())
datetime.datetime(2018, 6, 8, 0, 6, 58)
```

10.2.6 tzutc examples

```
>>> from datetime import *
>>> from dateutil import tz

>>> datetime.now()
datetime.datetime(2003, 9, 27, 9, 40, 1, 521290)

>>> datetime.now(tz.UTC)
datetime.datetime(2003, 9, 27, 12, 40, 12, 156379, tzinfo=tzutc())

>>> datetime.now(tz.UTC).tzname()
'UTC'
```

10.2.7 tzoffset examples

```
>>> from datetime import *
>>> from dateutil.tz import *

>>> datetime.now(tzoffset("BRST", -10800))
datetime.datetime(2003, 9, 27, 9, 52, 43, 624904,
          tzinfo=tzinfo=tzoffset('BRST', -10800))

>>> datetime.now(tzoffset("BRST", -10800)).tzname()
'BRST'

>>> datetime.now(tzoffset("BRST", -10800)).astimezone(UTC)
datetime.datetime(2003, 9, 27, 12, 53, 11, 446419,
          tzinfo=tzutc())
```

10.2.8 tzlocal examples

```
>>> from datetime import *
>>> from dateutil.tz import *

>>> datetime.now(tzlocal())
datetime.datetime(2003, 9, 27, 10, 1, 43, 673605,
          tzinfo=tzlocal())

>>> datetime.now(tzlocal()).tzname()
'BRST'

>>> datetime.now(tzlocal()).astimezone(tzoffset(None, 0))
datetime.datetime(2003, 9, 27, 13, 3, 0, 11493,
          tzinfo=tzoffset(None, 0))
```

10.2.9 tzstr examples

Here are examples of the recognized formats:

- *EST5EDT*
- *EST5EDT4,M4.1.0/02:00:00,M10-5-0/02:00*
- *EST5EDT4,95/02:00:00,298/02:00*
- *EST5EDT4,J96/02:00:00,J299/02:00*

Notice that if daylight information is not present, but a daylight abbreviation was provided, *tzstr* will follow the convention of using the first sunday of April to start daylight saving, and the last sunday of October to end it. If start or end time is not present, 2AM will be used, and if the daylight offset is not present, the standard offset plus one hour will be used. This convention is the same as used in the GNU libc.

This also means that some of the above examples are exactly equivalent, and all of these examples are equivalent in the year of 2003.

Here is the example mentioned in the

[https://docs.python.org/3/library/time.html time module documentation].

```
>>> os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'02:07:36 05/08/03 EDT'
>>> os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5.0'
>>> time.tzset()
>>> time.strftime('%X %x %Z')
'16:08:12 05/08/03 AEST'
```

And here is an example showing the same information using *tzstr*, without touching system settings.

```
>>> tz1 = tzstr('EST+05EDT,M4.1.0,M10.5.0')
>>> tz2 = tzstr('AEST-10AEDT-11,M10.5.0,M3.5.0')
>>> dt = datetime(2003, 5, 8, 2, 7, 36, tzinfo=tz1)
>>> dt.strftime('%X %x %Z')
'02:07:36 05/08/03 EDT'
>>> dt.astimezone(tz2).strftime('%X %x %Z')
'16:07:36 05/08/03 AEST'
```

Are these really equivalent?

```
>>> tzstr('EST5EDT') == tzstr('EST5EDT,M4.1.0,M10.5.0')
True
```

Check the daylight limit.

```
>>> tz = tzstr('EST+05EDT,M4.1.0,M10.5.0')
>>> datetime(2003, 4, 6, 1, 59, tzinfo=tz).tzname()
'EST'
>>> datetime(2003, 4, 6, 2, 00, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 0, 59, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 2, 00, tzinfo=tz).tzname()
'EST'
```

10.2.10 tzrange examples

```
>>> tzstr('EST5EDT') == tzrange("EST", -18000, "EDT")
True

>>> from dateutil.relativedelta import *
>>> range1 = tzrange("EST", -18000, "EDT")
>>> range2 = tzrange("EST", -18000, "EDT", -14400,
...                  relativedelta(hours=+2, month=4, day=1,
...                                weekday=SU(+1)),
...                  relativedelta(hours=+1, month=10, day=31,
...                                weekday=SU(-1)))
>>> tzstr('EST5EDT') == range1 == range2
True
```

Notice a minor detail in the last example: while the DST should end at 2AM, the delta will catch 1AM. That's because the daylight saving time should end at 2AM standard time (the difference between STD and DST is 1h in the given example) instead of the DST time. That's how the *tzinfo* subtypes should deal with the extra hour that happens when going back to the standard time. Check

[https://docs.python.org/3/library/datetime.html#datetime.tzinfo tzinfo documentation]

for more information.

10.2.11 tzfile examples

```
>>> tz = tzfile("/etc/localtime")
>>> datetime.now(tz)
datetime.datetime(2003, 9, 27, 12, 3, 48, 392138,
          tzinfo=tzfile('/etc/localtime'))

>>> datetime.now(tz).astimezone(UTC)
datetime.datetime(2003, 9, 27, 15, 3, 53, 70863,
          tzinfo=tzutc())

>>> datetime.now(tz).tzname()
'BRST'
>>> datetime(2003, 1, 1, tzinfo=tz).tzname()
'BRDT'
```

Check the daylight limit.

```
>>> tz = tzfile('/usr/share/zoneinfo/EST5EDT')
>>> datetime(2003, 4, 6, 1, 59, tzinfo=tz).tzname()
'EST'
>>> datetime(2003, 4, 6, 2, 00, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 0, 59, tzinfo=tz).tzname()
'EDT'
>>> datetime(2003, 10, 26, 1, 00, tzinfo=tz).tzname()
'EST'
```

10.2.12 tzical examples

Here is a sample file extracted from the RFC. This file defines the *EST5EDT* timezone, and will be used in the following example.

```
BEGIN:VTIMEZONE
TZID:US-Eastern
LAST-MODIFIED:19870101T000000Z
TZURL:http://zones.stds_r_us.net/tz/US-Eastern
BEGIN:STANDARD
DTSTART:19671029T020000
RRULE:FREQ=YEARLY;BYDAY=-1SU;BYMONTH=10
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
TZNAME:EST
END:STANDARD
BEGIN:DAYLIGHT
DTSTART:19870405T020000
RRULE:FREQ=YEARLY;BYDAY=1SU;BYMONTH=4
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
TZNAME:EDT
```

(continues on next page)

(continued from previous page)

```
END:DAYLIGHT
END:VTIMEZONE
```

And here is an example exploring a *tzical* type:

```
>>> from dateutil.tz import *; from datetime import *

>>> tz = tzical('samples/EST5EDT.ics')
>>> tz.keys()
['US-Eastern']

>>> est = tz.get('US-Eastern')
>>> est
<tzicalvtz 'US-Eastern'>

>>> datetime.now(est)
datetime.datetime(2003, 10, 6, 19, 44, 18, 667987,
          tzinfo=<tzicalvtz 'US-Eastern'>)

>>> est == tz.get()
True
```

Let's check the daylight ranges, as usual:

```
>>> datetime(2003, 4, 6, 1, 59, tzinfo=est).tzname()
'EST'
>>> datetime(2003, 4, 6, 2, 00, tzinfo=est).tzname()
'EDT'

>>> datetime(2003, 10, 26, 0, 59, tzinfo=est).tzname()
'EDT'
>>> datetime(2003, 10, 26, 1, 00, tzinfo=est).tzname()
'EST'
```

10.2.13 tzwin examples

```
>>> tz = tzwin("E. South America Standard Time")
```

10.2.14 tzwinlocal examples

```
>>> tz = tzwinlocal()
```

vim:ts=4:sw=4:et

10.3 Exercises

It is often useful to work through some examples in order to understand how a module works; on this page, there are several exercises of varying difficulty that you can use to learn how to use `dateutil`.

If you are interested in helping improve the documentation of `dateutil`, it is recommended that you attempt to complete these exercises with no resources *other than dateutil's documentation*. If you find that the documentation

is not clear enough to allow you to complete these exercises, open an issue on the dateutil issue tracker to let the developers know what part of the documentation needs improvement.

Table of Contents

10.3.1 Martin Luther King Day

Martin Luther King, Jr Day is a US holiday that occurs every year on the third Monday in January?

How would you generate a recurrence rule that generates Martin Luther King Day, starting from its first observance in 1986?

Test Script

To solve this exercise, copy-paste this script into a document, change anything between the `--- YOUR CODE ---` comment blocks.

```
# ------- YOUR CODE -------------#
from dateutil import rrule

MLK_DAY = <<YOUR CODE HERE>>

# -------------------------------#

from datetime import datetime
MLK_TEST_CASES = [
    ((datetime(1970, 1, 1), datetime(1980, 1, 1)),
     []),
    ((datetime(1980, 1, 1), datetime(1989, 1, 1)),
     [datetime(1986, 1, 20),
      datetime(1987, 1, 19),
      datetime(1988, 1, 18)]),
    ((datetime(2017, 2, 1), datetime(2022, 2, 1)),
     [datetime(2018, 1, 15, 0, 0),
      datetime(2019, 1, 21, 0, 0),
      datetime(2020, 1, 20, 0, 0),
      datetime(2021, 1, 18, 0, 0),
      datetime(2022, 1, 17, 0, 0)]
     ),
]

def test_mlk_day():
    for (between_args, expected) in MLK_TEST_CASES:
        assert MLK_DAY.between(*between_args) == expected

if __name__ == "__main__":
    test_mlk_day()
    print('Success!')
```

A solution to this problem is provided here.

10.3.2 Next Monday meeting

A team has a meeting at 10 AM every Monday and wants a function that tells them, given a `datetime.datetime` object, what is the date and time of the *next* Monday meeting? This is probably best accomplished using a relativedelta.

Test Script

To solve this exercise, copy-paste this script into a document, change anything between the `--- YOUR CODE ---` comment blocks.

```
# --------- YOUR CODE -------------- #
from dateutil import relativedelta

def next_monday(dt):
    <<YOUR CODE HERE>>

# ---------------------------------- #

from datetime import datetime
from dateutil import tz

NEXT_MONDAY_CASES = [
    (datetime(2018, 4, 11, 14, 30, 15, 123456),
     datetime(2018, 4, 16, 10, 0)),
    (datetime(2018, 4, 16, 10, 0),
     datetime(2018, 4, 16, 10, 0)),
    (datetime(2018, 4, 16, 10, 30),
     datetime(2018, 4, 23, 10, 0)),
    (datetime(2018, 4, 14, 9, 30, tzinfo=tz.gettz('America/New_York')),
     datetime(2018, 4, 16, 10, 0, tzinfo=tz.gettz('America/New_York'))),
]

def test_next_monday_1():
    for dt_in, dt_out in NEXT_MONDAY_CASES:
        assert next_monday(dt_in) == dt_out

if __name__ == "__main__":
    test_next_monday_1()
    print('Success!')
```

10.3.3 Parsing a local tzname

Three-character time zone abbreviations are *not* unique in that they do not explicitly map to a time zone. A list of time zone abbreviations in use can be found here. This means that parsing a datetime string such as `'2018-01-01 12:30:30 CST'` is ambiguous without context. Using dateutil.parser and dateutil.tz, it is possible to provide a context such that these local names are converted to proper time zones.

Problem 1

Given the context that you will only be parsing dates coming from the continental United States, India and Japan, write a function that parses a datetime string and returns a timezone-aware `datetime` with an IANA-style timezone attached.

Note: For the purposes of the experiment, you may ignore the portions of the United States like Arizona and parts of Indiana that do not observe daylight saving time.

Test Script

To solve this exercise, copy-paste this script into a document, change anything between the `--- YOUR CODE ---` comment blocks.

```
# --------- YOUR CODE -------------- #
from dateutil.parser import parse
from dateutil import tz

def parse_func_us_jp_ind():
    <<YOUR CODE HERE>>

# ---------------------------------- #

from dateutil import tz
from datetime import datetime


PARSE_TZ_TEST_DATETIMES = [
    datetime(2018, 1, 1, 12, 0),
    datetime(2018, 3, 20, 2, 0),
    datetime(2018, 5, 12, 3, 30),
    datetime(2014, 9, 1, 23)
]

PARSE_TZ_TEST_ZONES = [
    tz.gettz('America/New_York'),
    tz.gettz('America/Chicago'),
    tz.gettz('America/Denver'),
    tz.gettz('America/Los_Angeles'),
    tz.gettz('Asia/Kolkata'),
    tz.gettz('Asia/Tokyo'),
]

def test_parse():
    for tzi in PARSE_TZ_TEST_ZONES:
        for dt in PARSE_TZ_TEST_DATETIMES:
            dt_exp = dt.replace(tzinfo=tzi)
            dtstr = dt_exp.strftime('%Y-%m-%d %H:%M:%S %Z')

            dt_act = parse_func_us_jp_ind(dtstr)
            assert dt_act == dt_exp
            assert dt_act.tzinfo is dt_exp.tzinfo

if __name__ == "__main__":
    test_parse()
    print('Success!')
```

Problem 2

Given the context that you will *only* be passed dates from India or Ireland, write a function that correctly parses all *unambiguous* time zone strings to aware datetimes localized to the correct IANA zone, and for *ambiguous* time zone strings default to India.

Test Script

To solve this exercise, copy-paste this script into a document, change anything between the `--- YOUR CODE ---` comment blocks.

```
# --------- YOUR CODE -------------- #
from dateutil.parser import parse
from dateutil import tz

def parse_func_ind_ire():
    <<YOUR CODE HERE>>

# ---------------------------------- #
ISRAEL = tz.gettz('Asia/Jerusalem')
INDIA = tz.gettz('Asia/Kolkata')
PARSE_IXT_TEST_CASE = [
    ('2018-02-03 12:00 IST+02:00', datetime(2018, 2, 3, 12, tzinfo=ISRAEL)),
    ('2018-06-14 12:00 IDT+03:00', datetime(2018, 6, 14, 12, tzinfo=ISRAEL)),
    ('2018-06-14 12:00 IST', datetime(2018, 6, 14, 12, tzinfo=INDIA)),
    ('2018-06-14 12:00 IST+05:30', datetime(2018, 6, 14, 12, tzinfo=INDIA)),
    ('2018-02-03 12:00 IST', datetime(2018, 2, 3, 12, tzinfo=INDIA)),
]


def test_parse_ixt():
    for dtstr, dt_exp in PARSE_IXT_TEST_CASE:
        dt_act = parse_func_ind_ire(dtstr)
        assert dt_act == dt_exp, (dt_act, dt_exp)
        assert dt_act.tzinfo is dt_exp.tzinfo, (dt_act, dt_exp)

if __name__ == "__main__":
    test_parse_ixt()
    print('Success!')
```

10.4 easter

This module offers a generic easter computing method for any given year, using Western, Orthodox or Julian algorithms.

`dateutil.easter.`**`easter`**(*year*, *method=3*)

This method was ported from the work done by GM Arts, on top of the algorithm by Claus Tondering, which was based in part on the algorithm of Ouding (1940), as quoted in "Explanatory Supplement to the Astronomical Almanac", P. Kenneth Seidelmann, editor.

This algorithm implements three different easter calculation methods:

1 - Original calculation in Julian calendar, valid in dates after 326 AD

2 - Original method, with date converted to Gregorian calendar, valid in years 1583 to 4099

3 - Revised method, in Gregorian calendar, valid in years 1583 to 4099 as well

These methods are represented by the constants:

- `EASTER_JULIAN = 1`
- `EASTER_ORTHODOX = 2`
- `EASTER_WESTERN = 3`

The default method is method 3.

More about the algorithm may be found at:

GM Arts: Easter Algorithms

and

The Calendar FAQ: Easter

10.5 parser

This module offers a generic date/time string parser which is able to parse most known formats to represent a date and/or time.

This module attempts to be forgiving with regards to unlikely input formats, returning a datetime object even for dates which are ambiguous. If an element of a date/time stamp is omitted, the following rules are applied:

- If AM or PM is left unspecified, a 24-hour clock is assumed, however, an hour on a 12-hour clock (`0 <= hour <= 12`) *must* be specified if AM or PM is specified.
- If a time zone is omitted, a timezone-naive datetime is returned.

If any other elements are missing, they are taken from the `datetime.datetime` object passed to the parameter `default`. If this results in a day number exceeding the valid number of days per month, the value falls back to the end of the month.

Additional resources about date/time string formats can be found below:

- A summary of the international standard date and time notation
- W3C Date and Time Formats
- Time Formats (Planetary Rings Node)
- CPAN ParseDate module
- Java SimpleDateFormat Class

`parser.`**`parse`**(*parserinfo=None*, ***kwargs*)

Parse a string in one of the supported formats, using the `parserinfo` parameters.

Parameters

- **`timestr`** – A string containing a date/time stamp.
- **`parserinfo`** – A *`parserinfo`* object containing parameters for the parser. If `None`, the default arguments to the *`parserinfo`* constructor are used.

The `**kwargs` parameter takes the following keyword arguments:

Parameters

- **`default`** – The default datetime object, if this is a datetime object and not `None`, elements specified in `timestr` replace elements in the default object.
- **`ignoretz`** – If set `True`, time zones in parsed strings are ignored and a naive `datetime` object is returned.
- **`tzinfos`** – Additional time zone names / aliases which may be present in the string. This argument maps time zone names (and optionally offsets from those time zones) to time zones. This parameter can be a dictionary with timezone aliases mapping time zone names to time zones or a function taking two parameters (`tzname` and `tzoffset`) and returning a time zone.

The timezones to which the names are mapped can be an integer offset from UTC in seconds or a `tzinfo` object.

```
>>> from dateutil.parser import parse
>>> from dateutil.tz import gettz
>>> tzinfos = {"BRST": -7200, "CST": gettz("America/Chicago")}
>>> parse("2012-01-19 17:21:00 BRST", tzinfos=tzinfos)
datetime.datetime(2012, 1, 19, 17, 21, tzinfo=tzoffset(u'BRST', -
↪7200))
>>> parse("2012-01-19 17:21:00 CST", tzinfos=tzinfos)
datetime.datetime(2012, 1, 19, 17, 21,
                  tzinfo=tzfile('/usr/share/zoneinfo/America/
↪Chicago'))
```

This parameter is ignored if `ignoretz` is set.

- **dayfirst** – Whether to interpret the first value in an ambiguous 3-integer date (e.g. 01/05/09) as the day (`True`) or month (`False`). If `yearfirst` is set to `True`, this distinguishes between YDM and YMD. If set to `None`, this value is retrieved from the current *parserinfo* object (which itself defaults to `False`).
- **yearfirst** – Whether to interpret the first value in an ambiguous 3-integer date (e.g. 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. If this is set to `None`, the value is retrieved from the current *parserinfo* object (which itself defaults to `False`).
- **fuzzy** – Whether to allow fuzzy parsing, allowing for string like "Today is January 1, 2047 at 8:21:00AM".
- **fuzzy_with_tokens** – If `True`, `fuzzy` is automatically set to True, and the parser will return a tuple where the first element is the parsed `datetime.datetime` datetimestamp and the second element is a tuple containing the portions of the string which were ignored:

```
>>> from dateutil.parser import parse
>>> parse("Today is January 1, 2047 at 8:21:00AM", fuzzy_with_
↪tokens=True)
(datetime.datetime(2047, 1, 1, 8, 21), (u'Today is ', u' ', u'at
↪'))
```

Returns Returns a `datetime.datetime` object or, if the `fuzzy_with_tokens` option is `True`, returns a tuple, the first element being a `datetime.datetime` object, the second a tuple containing the fuzzy tokens.

Raises

- **ValueError** – Raised for invalid or unknown string format, if the provided `tzinfo` is not in a valid format, or if an invalid date would be created.
- **OverflowError** – Raised if the parsed date exceeds the largest valid C integer on your system.

class dateutil.parser.**parserinfo**(*dayfirst=False*, *yearfirst=False*)

Class which handles what inputs are accepted. Subclass this to customize the language and acceptable values for each parameter.

Parameters

- **dayfirst** – Whether to interpret the first value in an ambiguous 3-integer date (e.g. 01/05/09) as the day (`True`) or month (`False`). If `yearfirst` is set to `True`, this distinguishes between YDM and YMD. Default is `False`.

- **yearfirst** – Whether to interpret the first value in an ambiguous 3-integer date (e.g. 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. Default is `False`.

AMPM = [('am', 'a'), ('pm', 'p')]

HMS = [('h', 'hour', 'hours'), ('m', 'minute', 'minutes'), ('s', 'second', 'seconds')]

JUMP = [' ', '.', ',', ';', '-', '/', "'", 'at', 'on', 'and', 'ad', 'm', 't', 'of', 's

MONTHS = [('Jan', 'January'), ('Feb', 'February'), ('Mar', 'March'), ('Apr', 'April'),

PERTAIN = ['of']

TZOFFSET = {}

UTCZONE = ['UTC', 'GMT', 'Z', 'z']

WEEKDAYS = [('Mon', 'Monday'), ('Tue', 'Tuesday'), ('Wed', 'Wednesday'), ('Thu', 'Thur

ampm(*name*)

convertyear(*year*, *century_specified=False*)
Converts two-digit years to year within [-50, 49] range of self._year (current local time)

hms(*name*)

jump(*name*)

month(*name*)

pertain(*name*)

tzoffset(*name*)

utczone(*name*)

validate(*res*)

weekday(*name*)

classmethod parser.**isoparse**(*dt_str*)
Parse an ISO-8601 datetime string into a `datetime.datetime`.

An ISO-8601 datetime string consists of a date portion, followed optionally by a time portion - the date and time portions are separated by a single character separator, which is `T` in the official standard. Incomplete date formats (such as `YYYY-MM`) may *not* be combined with a time portion.

Supported date formats are:

Common:

- `YYYY`
- `YYYY-MM` or `YYYYMM`
- `YYYY-MM-DD` or `YYYYMMDD`

Uncommon:

- `YYYY-Www` or `YYYYWww` - ISO week (day defaults to 0)
- `YYYY-Www-D` or `YYYYWwwD` - ISO week and day

The ISO week and day numbering follows the same logic as `datetime.date.isocalendar()`.

Supported time formats are:

- `hh`

- `hh:mm` or `hhmm`
- `hh:mm:ss` or `hhmmss`
- `hh:mm:ss.ssssss` (Up to 6 sub-second digits)

Midnight is a special case for *hh*, as the standard supports both 00:00 and 24:00 as a representation. The decimal separator can be either a dot or a comma.

Caution: Support for fractional components other than seconds is part of the ISO-8601 standard, but is not currently implemented in this parser.

Supported time zone offset formats are:

- *Z* (UTC)
- *±HH:MM*
- *±HHMM*
- *±HH*

Offsets will be represented as *dateutil.tz.tzoffset* objects, with the exception of UTC, which will be represented as *dateutil.tz.tzutc*. Time zone offsets equivalent to UTC (such as *+00:00*) will also be represented as *dateutil.tz.tzutc*.

Parameters **dt_str** – A string or stream containing only an ISO-8601 datetime string

Returns Returns a `datetime.datetime` representing the string. Unspecified components default to their lowest value.

Warning: As of version 2.7.0, the strictness of the parser should not be considered a stable part of the contract. Any valid ISO-8601 string that parses correctly with the default settings will continue to parse correctly in future versions, but invalid strings that currently fail (e.g. `2017-01-01T00:00+00:00:00`) are not guaranteed to continue failing in future versions if they encode a valid date.

New in version 2.7.0.

10.6 relativedelta

class dateutil.relativedelta.**relativedelta**(*dt1=None*, *dt2=None*, *years=0*, *months=0*, *days=0*, *leapdays=0*, *weeks=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*, *year=None*, *month=None*, *day=None*, *weekday=None*, *yearday=None*, *nlyearday=None*, *hour=None*, *minute=None*, *second=None*, *microsecond=None*)

The relativedelta type is designed to be applied to an existing datetime and can replace specific components of that datetime, or represents an interval of time.

It is based on the specification of the excellent work done by M.-A. Lemburg in his mx.DateTime extension. However, notice that this type does *NOT* implement the same algorithm as his work. Do *NOT* expect it to behave like mx.DateTime's counterpart.

There are two different ways to build a relativedelta instance. The first one is passing it two date/datetime classes:

```
relativedelta(datetime1, datetime2)
```

The second one is passing it any number of the following keyword arguments:

```
relativedelta(arg1=x,arg2=y,arg3=z...)

year, month, day, hour, minute, second, microsecond:
    Absolute information (argument is singular); adding or subtracting a
    relativedelta with absolute information does not perform an arithmetic
    operation, but rather REPLACES the corresponding value in the
    original datetime with the value(s) in relativedelta.

years, months, weeks, days, hours, minutes, seconds, microseconds:
    Relative information, may be negative (argument is plural); adding
    or subtracting a relativedelta with relative information performs
    the corresponding arithmetic operation on the original datetime value
    with the information in the relativedelta.

weekday:
    One of the weekday instances (MO, TU, etc) available in the
    relativedelta module. These instances may receive a parameter N,
    specifying the Nth weekday, which could be positive or negative
    (like MO(+1) or MO(-2)). Not specifying it is the same as specifying
    +1. You can also use an integer, where 0=MO. This argument is always
    relative e.g. if the calculated date is already Monday, using MO(1)
    or MO(-1) won't change the day. To effectively make it absolute, use
    it in combination with the day argument (e.g. day=1, MO(1) for first
    Monday of the month).

leapdays:
    Will add given days to the date found, if year is a leap
    year, and the date found is post 28 of february.

yearday, nlyearday:
    Set the yearday or the non-leap year day (jump leap days).
    These are converted to day/month/leapdays information.
```

There are relative and absolute forms of the keyword arguments. The plural is relative, and the singular is absolute. For each argument in the order below, the absolute form is applied first (by setting each attribute to that value) and then the relative form (by adding the value to the attribute).

The order of attributes considered when this relativedelta is added to a datetime is:

1. Year
2. Month
3. Day
4. Hours
5. Minutes
6. Seconds
7. Microseconds

Finally, weekday is applied, using the rule described above.

For example

```
>>> from datetime import datetime
>>> from dateutil.relativedelta import relativedelta, MO
>>> dt = datetime(2018, 4, 9, 13, 37, 0)
>>> delta = relativedelta(hours=25, day=1, weekday=MO(1))
>>> dt + delta
datetime.datetime(2018, 4, 2, 14, 37)
```

First, the day is set to 1 (the first of the month), then 25 hours are added, to get to the 2nd day and 14th hour, finally the weekday is applied, but since the 2nd is already a Monday there is no effect.

normalized()

Return a version of this object represented entirely using integer values for the relative attributes.

```
>>> relativedelta(days=1.5, hours=2).normalized()
relativedelta(days=+1, hours=+14)
```

Returns Returns a *dateutil.relativedelta.relativedelta* object.

weeks

10.6.1 Examples

```
>>> from datetime import *; from dateutil.relativedelta import *
>>> import calendar
>>> NOW = datetime(2003, 9, 17, 20, 54, 47, 282310)
>>> TODAY = date(2003, 9, 17)
```

Let's begin our trip:

```
>>> from datetime import *; from dateutil.relativedelta import *
>>> import calendar
```

Store some values:

```
>>> NOW = datetime.now()
>>> TODAY = date.today()
>>> NOW
datetime.datetime(2003, 9, 17, 20, 54, 47, 282310)
>>> TODAY
datetime.date(2003, 9, 17)
```

Next month

```
>>> NOW+relativedelta(months=+1)
datetime.datetime(2003, 10, 17, 20, 54, 47, 282310)
```

Next month, plus one week.

```
>>> NOW+relativedelta(months=+1, weeks=+1)
datetime.datetime(2003, 10, 24, 20, 54, 47, 282310)
```

Next month, plus one week, at 10am.

```
>>> TODAY+relativedelta(months=+1, weeks=+1, hour=10)
datetime.datetime(2003, 10, 24, 10, 0)
```

Here is another example using an absolute relativedelta. Notice the use of year and month (both singular) which causes the values to be *replaced* in the original datetime rather than performing an arithmetic operation on them.

```
>>> NOW+relativedelta(year=1, month=1)
datetime.datetime(1, 1, 17, 20, 54, 47, 282310)
```

Let's try the other way around. Notice that the hour setting we get in the relativedelta is relative, since it's a difference, and the weeks parameter has gone.

```
>>> relativedelta(datetime(2003, 10, 24, 10, 0), TODAY)
relativedelta(months=+1, days=+7, hours=+10)
```

One month before one year.

```
>>> NOW+relativedelta(years=+1, months=-1)
datetime.datetime(2004, 8, 17, 20, 54, 47, 282310)
```

How does it handle months with different numbers of days? Notice that adding one month will never cross the month boundary.

```
>>> date(2003,1,27)+relativedelta(months=+1)
datetime.date(2003, 2, 27)
>>> date(2003,1,31)+relativedelta(months=+1)
datetime.date(2003, 2, 28)
>>> date(2003,1,31)+relativedelta(months=+2)
datetime.date(2003, 3, 31)
```

The logic for years is the same, even on leap years.

```
>>> date(2000,2,28)+relativedelta(years=+1)
datetime.date(2001, 2, 28)
>>> date(2000,2,29)+relativedelta(years=+1)
datetime.date(2001, 2, 28)

>>> date(1999,2,28)+relativedelta(years=+1)
datetime.date(2000, 2, 28)
>>> date(1999,3,1)+relativedelta(years=+1)
datetime.date(2000, 3, 1)

>>> date(2001,2,28)+relativedelta(years=-1)
datetime.date(2000, 2, 28)
>>> date(2001,3,1)+relativedelta(years=-1)
datetime.date(2000, 3, 1)
```

Next friday

```
>>> TODAY+relativedelta(weekday=FR)
datetime.date(2003, 9, 19)

>>> TODAY+relativedelta(weekday=calendar.FRIDAY)
datetime.date(2003, 9, 19)
```

Last friday in this month.

```
>>> TODAY+relativedelta(day=31, weekday=FR(-1))
datetime.date(2003, 9, 26)
```

Next wednesday (it's today!).

```
>>> TODAY+relativedelta(weekday=WE(+1))
datetime.date(2003, 9, 17)
```

Next wednesday, but not today.

```
>>> TODAY+relativedelta(days=+1, weekday=WE(+1))
datetime.date(2003, 9, 24)
```

Following ISO year week number notation find the first day of the 15th week of 1997.

```
>>> datetime(1997,1,1)+relativedelta(day=4, weekday=MO(-1), weeks=+14)
datetime.datetime(1997, 4, 7, 0, 0)
```

How long ago has the millennium changed?

```
>>> relativedelta(NOW, date(2001,1,1))
relativedelta(years=+2, months=+8, days=+16,
              hours=+20, minutes=+54, seconds=+47, microseconds=+282310)
```

How old is John?

```
>>> johnbirthday = datetime(1978, 4, 5, 12, 0)
>>> relativedelta(NOW, johnbirthday)
relativedelta(years=+25, months=+5, days=+12,
          hours=+8, minutes=+54, seconds=+47, microseconds=+282310)
```

It works with dates too.

```
>>> relativedelta(TODAY, johnbirthday)
relativedelta(years=+25, months=+5, days=+11, hours=+12)
```

Obtain today's date using the yearday:

```
>>> date(2003, 1, 1)+relativedelta(yearday=260)
datetime.date(2003, 9, 17)
```

We can use today's date, since yearday should be absolute in the given year:

```
>>> TODAY+relativedelta(yearday=260)
datetime.date(2003, 9, 17)
```

Last year it should be in the same day:

```
>>> date(2002, 1, 1)+relativedelta(yearday=260)
datetime.date(2002, 9, 17)
```

But not in a leap year:

```
>>> date(2000, 1, 1)+relativedelta(yearday=260)
datetime.date(2000, 9, 16)
```

We can use the non-leap year day to ignore this:

```
>>> date(2000, 1, 1)+relativedelta(nlyearday=260)
datetime.date(2000, 9, 17)
```

10.7 rrule

The rrule module offers a small, complete, and very fast, implementation of the recurrence rules documented in the iCalendar RFC, including support for caching of results.

10.7.1 Classes

class dateutil.rrule.**rrule**(*freq*, *dtstart=None*, *interval=1*, *wkst=None*, *count=None*, *until=None*, *bysetpos=None*, *bymonth=None*, *bymonthday=None*, *byyearday=None*, *byeaster=None*, *byweekno=None*, *byweekday=None*, *byhour=None*, *byminute=None*, *bysecond=None*, *cache=False*)

That's the base of the rrule operation. It accepts all the keywords defined in the RFC as its constructor parameters (except byday, which was renamed to byweekday) and more. The constructor prototype is:

```
rrule(freq)
```

Where freq must be one of YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, or SECONDLY.

Note: Per RFC section 3.3.10, recurrence instances falling on invalid dates and times are ignored rather than coerced:

> Recurrence rules may generate recurrence instances with an invalid date (e.g., February 30) or nonexistent local time (e.g., 1:30 AM on a day where the local time is moved forward by an hour at 1:00 AM). Such recurrence instances MUST be ignored and MUST NOT be counted as part of the recurrence set.

This can lead to possibly surprising behavior when, for example, the start date occurs at the end of the month:

```
>>> from dateutil.rrule import rrule, MONTHLY
>>> from datetime import datetime
>>> start_date = datetime(2014, 12, 31)
>>> list(rrule(freq=MONTHLY, count=4, dtstart=start_date))
... # doctest: +NORMALIZE_WHITESPACE
[datetime.datetime(2014, 12, 31, 0, 0),
 datetime.datetime(2015, 1, 31, 0, 0),
 datetime.datetime(2015, 3, 31, 0, 0),
 datetime.datetime(2015, 5, 31, 0, 0)]
```

Additionally, it supports the following keyword arguments:

Parameters

- **dtstart** – The recurrence start. Besides being the base for the recurrence, missing parameters in the final recurrence instances will also be extracted from this date. If not given, datetime.now() will be used instead.
- **interval** – The interval between each freq iteration. For example, when using YEARLY, an interval of 2 means once every two years, but with HOURLY, it means once every two hours. The default interval is 1.
- **wkst** – The week start day. Must be one of the MO, TU, WE constants, or an integer, specifying the first day of the week. This will affect recurrences based on weekly periods. The default week start is got from calendar.firstweekday(), and may be modified by calendar.setfirstweekday().

- **count** – If given, this determines how many occurrences will be generated.

  Note: As of version 2.5.0, the use of the keyword `until` in conjunction with `count` is deprecated, to make sure `dateutil` is fully compliant with RFC-5545 Sec. 3.3.10. Therefore, `until` and `count` **must not** occur in the same call to `rrule`.

- **until** – If given, this must be a datetime instance specifying the upper-bound limit of the recurrence. The last recurrence in the rule is the greatest datetime that is less than or equal to the value specified in the `until` parameter.

  Note: As of version 2.5.0, the use of the keyword `until` in conjunction with `count` is deprecated, to make sure `dateutil` is fully compliant with RFC-5545 Sec. 3.3.10. Therefore, `until` and `count` **must not** occur in the same call to `rrule`.

- **bysetpos** – If given, it must be either an integer, or a sequence of integers, positive or negative. Each given integer will specify an occurrence number, corresponding to the nth occurrence of the rule inside the frequency period. For example, a bysetpos of -1 if combined with a MONTHLY frequency, and a byweekday of (MO, TU, WE, TH, FR), will result in the last work day of every month.
- **bymonth** – If given, it must be either an integer, or a sequence of integers, meaning the months to apply the recurrence to.
- **bymonthday** – If given, it must be either an integer, or a sequence of integers, meaning the month days to apply the recurrence to.
- **byyearday** – If given, it must be either an integer, or a sequence of integers, meaning the year days to apply the recurrence to.
- **byeaster** – If given, it must be either an integer, or a sequence of integers, positive or negative. Each integer will define an offset from the Easter Sunday. Passing the offset 0 to byeaster will yield the Easter Sunday itself. This is an extension to the RFC specification.
- **byweekno** – If given, it must be either an integer, or a sequence of integers, meaning the week numbers to apply the recurrence to. Week numbers have the meaning described in ISO8601, that is, the first week of the year is that containing at least four days of the new year.
- **byweekday** – If given, it must be either an integer (0 == MO), a sequence of integers, one of the weekday constants (MO, TU, etc), or a sequence of these constants. When given, these variables will define the weekdays where the recurrence will be applied. It's also possible to use an argument n for the weekday instances, which will mean the nth occurrence of this weekday in the period. For example, with MONTHLY, or with YEARLY and BYMONTH, using FR(+1) in byweekday will specify the first friday of the month where the recurrence happens. Notice that in the RFC documentation, this is specified as BYDAY, but was renamed to avoid the ambiguity of that keyword.
- **byhour** – If given, it must be either an integer, or a sequence of integers, meaning the hours to apply the recurrence to.
- **byminute** – If given, it must be either an integer, or a sequence of integers, meaning the minutes to apply the recurrence to.
- **bysecond** – If given, it must be either an integer, or a sequence of integers, meaning the seconds to apply the recurrence to.

- **cache** – If given, it must be a boolean value specifying to enable or disable caching of results. If you will use the same rrule instance multiple times, enabling caching will improve the performance considerably.

class dateutil.rrule.**rruleset**(*cache=False*)

The rruleset type allows more complex recurrence setups, mixing multiple rules, dates, exclusion rules, and exclusion dates. The type constructor takes the following keyword arguments:

Parameters **cache** – If True, caching of results will be enabled, improving performance of multiple queries considerably.

10.7.2 Functions

dateutil.rrule.**rrulestr**(*s*, ***kwargs*)

Parses a string representation of a recurrence rule or set of recurrence rules.

Parameters

- **s** – Required, a string defining one or more recurrence rules.
- **dtstart** – If given, used as the default recurrence start if not specified in the rule string.
- **cache** – If set `True` caching of results will be enabled, improving performance of multiple queries considerably.
- **unfold** – If set `True` indicates that a rule string is split over more than one line and should be joined before processing.
- **forceset** – If set `True` forces a *dateutil.rrule.rruleset* to be returned.
- **compatible** – If set `True` forces `unfold` and `forceset` to be `True`.
- **ignoretz** – If set `True`, time zones in parsed strings are ignored and a naive `datetime.datetime` object is returned.
- **tzids** – If given, a callable or mapping used to retrieve a `datetime.tzinfo` from a string representation. Defaults to *dateutil.tz.gettz()*.
- **tzinfos** – Additional time zone names / aliases which may be present in a string representation. See *dateutil.parser.parse()* for more information.

Returns Returns a *dateutil.rrule.rruleset* or *dateutil.rrule.rrule*

10.7.3 rrule examples

These examples were converted from the RFC.

Prepare the environment.

```
>>> from dateutil.rrule import *
>>> from dateutil.parser import *
>>> from datetime import *

>>> import pprint
>>> import sys
>>> sys.displayhook = pprint.pprint
```

Daily, for 10 occurrences.

```
>>> list(rrule(DAILY, count=10,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 9, 6, 9, 0),
 datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 10, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0)]
```

Daily until December 24, 1997

```
>>> list(rrule(DAILY,
...            dtstart=parse("19970902T090000"),
...            until=parse("19971224T000000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 ...
 datetime.datetime(1997, 12, 21, 9, 0),
 datetime.datetime(1997, 12, 22, 9, 0),
 datetime.datetime(1997, 12, 23, 9, 0)]
```

Every other day, 5 occurrences.

```
>>> list(rrule(DAILY, interval=2, count=5,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 6, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0),
 datetime.datetime(1997, 9, 10, 9, 0)]
```

Every 10 days, 5 occurrences.

```
>>> list(rrule(DAILY, interval=10, count=5,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Everyday in January, for 3 years.

```
>>> list(rrule(YEARLY, bymonth=1, byweekday=range(7),
...            dtstart=parse("19980101T090000"),
...            until=parse("20000131T090000")))
[datetime.datetime(1998, 1, 1, 9, 0),
 datetime.datetime(1998, 1, 2, 9, 0),
 ...
 datetime.datetime(1998, 1, 30, 9, 0),
 datetime.datetime(1998, 1, 31, 9, 0),
 datetime.datetime(1999, 1, 1, 9, 0),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1999, 1, 2, 9, 0),
 ...
 datetime.datetime(1999, 1, 30, 9, 0),
 datetime.datetime(1999, 1, 31, 9, 0),
 datetime.datetime(2000, 1, 1, 9, 0),
 datetime.datetime(2000, 1, 2, 9, 0),
 ...
 datetime.datetime(2000, 1, 30, 9, 0),
 datetime.datetime(2000, 1, 31, 9, 0)]
```

Same thing, in another way.

```
>>> list(rrule(DAILY, bymonth=1,
...            dtstart=parse("19980101T090000"),
...            until=parse("20000131T090000")))
[datetime.datetime(1998, 1, 1, 9, 0),
 ...
 datetime.datetime(2000, 1, 31, 9, 0)]
```

Weekly for 10 occurrences.

```
>>> list(rrule(WEEKLY, count=10,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 7, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 21, 9, 0),
 datetime.datetime(1997, 10, 28, 9, 0),
 datetime.datetime(1997, 11, 4, 9, 0)]
```

Every other week, 6 occurrences.

```
>>> list(rrule(WEEKLY, interval=2, count=6,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 28, 9, 0),
 datetime.datetime(1997, 11, 11, 9, 0)]
```

Weekly on Tuesday and Thursday for 5 weeks.

```
>>> list(rrule(WEEKLY, count=10, wkst=SU, byweekday=(TU,TH),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 18, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1997, 9, 25, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0)]
```

Every other week on Tuesday and Thursday, for 8 occurrences.

```
>>> list(rrule(WEEKLY, interval=2, count=8,
...            wkst=SU, byweekday=(TU,TH),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 18, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 14, 9, 0),
 datetime.datetime(1997, 10, 16, 9, 0)]
```

Monthly on the 1st Friday for ten occurrences.

```
>>> list(rrule(MONTHLY, count=10, byweekday=FR(1),
...            dtstart=parse("19970905T090000")))
[datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 10, 3, 9, 0),
 datetime.datetime(1997, 11, 7, 9, 0),
 datetime.datetime(1997, 12, 5, 9, 0),
 datetime.datetime(1998, 1, 2, 9, 0),
 datetime.datetime(1998, 2, 6, 9, 0),
 datetime.datetime(1998, 3, 6, 9, 0),
 datetime.datetime(1998, 4, 3, 9, 0),
 datetime.datetime(1998, 5, 1, 9, 0),
 datetime.datetime(1998, 6, 5, 9, 0)]
```

Every other month on the 1st and last Sunday of the month for 10 occurrences.

```
>>> list(rrule(MONTHLY, interval=2, count=10,
...            byweekday=(SU(1), SU(-1)),
...            dtstart=parse("19970907T090000")))
[datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 28, 9, 0),
 datetime.datetime(1997, 11, 2, 9, 0),
 datetime.datetime(1997, 11, 30, 9, 0),
 datetime.datetime(1998, 1, 4, 9, 0),
 datetime.datetime(1998, 1, 25, 9, 0),
 datetime.datetime(1998, 3, 1, 9, 0),
 datetime.datetime(1998, 3, 29, 9, 0),
 datetime.datetime(1998, 5, 3, 9, 0),
 datetime.datetime(1998, 5, 31, 9, 0)]
```

Monthly on the second to last Monday of the month for 6 months.

```
>>> list(rrule(MONTHLY, count=6, byweekday=MO(-2),
...            dtstart=parse("19970922T090000")))
[datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 20, 9, 0),
 datetime.datetime(1997, 11, 17, 9, 0),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1997, 12, 22, 9, 0),
 datetime.datetime(1998, 1, 19, 9, 0),
 datetime.datetime(1998, 2, 16, 9, 0)]
```

Monthly on the third to the last day of the month, for 6 months.

```
>>> list(rrule(MONTHLY, count=6, bymonthday=-3,
...            dtstart=parse("19970928T090000")))
[datetime.datetime(1997, 9, 28, 9, 0),
 datetime.datetime(1997, 10, 29, 9, 0),
 datetime.datetime(1997, 11, 28, 9, 0),
 datetime.datetime(1997, 12, 29, 9, 0),
 datetime.datetime(1998, 1, 29, 9, 0),
 datetime.datetime(1998, 2, 26, 9, 0)]
```

Monthly on the 2nd and 15th of the month for 5 occurrences.

```
>>> list(rrule(MONTHLY, count=5, bymonthday=(2,15),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 15, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 15, 9, 0),
 datetime.datetime(1997, 11, 2, 9, 0)]
```

Monthly on the first and last day of the month for 3 occurrences.

```
>>> list(rrule(MONTHLY, count=5, bymonthday=(-1,1,),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 10, 1, 9, 0),
 datetime.datetime(1997, 10, 31, 9, 0),
 datetime.datetime(1997, 11, 1, 9, 0),
 datetime.datetime(1997, 11, 30, 9, 0)]
```

Every 18 months on the 10th thru 15th of the month for 10 occurrences.

```
>>> list(rrule(MONTHLY, interval=18, count=10,
...            bymonthday=range(10,16),
...            dtstart=parse("19970910T090000")))
[datetime.datetime(1997, 9, 10, 9, 0),
 datetime.datetime(1997, 9, 11, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 13, 9, 0),
 datetime.datetime(1997, 9, 14, 9, 0),
 datetime.datetime(1997, 9, 15, 9, 0),
 datetime.datetime(1999, 3, 10, 9, 0),
 datetime.datetime(1999, 3, 11, 9, 0),
 datetime.datetime(1999, 3, 12, 9, 0),
 datetime.datetime(1999, 3, 13, 9, 0)]
```

Every Tuesday, every other month, 6 occurrences.

```
>>> list(rrule(MONTHLY, interval=2, count=6, byweekday=TU,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 16, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0),
 datetime.datetime(1997, 9, 30, 9, 0),
 datetime.datetime(1997, 11, 4, 9, 0)]
```

Yearly in June and July for 10 occurrences.

```
>>> list(rrule(YEARLY, count=4, bymonth=(6,7),
...            dtstart=parse("19970610T090000")))
[datetime.datetime(1997, 6, 10, 9, 0),
 datetime.datetime(1997, 7, 10, 9, 0),
 datetime.datetime(1998, 6, 10, 9, 0),
 datetime.datetime(1998, 7, 10, 9, 0)]
```

Every 3rd year on the 1st, 100th and 200th day for 4 occurrences.

```
>>> list(rrule(YEARLY, count=4, interval=3, byyearday=(1,100,200),
...            dtstart=parse("19970101T090000")))
[datetime.datetime(1997, 1, 1, 9, 0),
 datetime.datetime(1997, 4, 10, 9, 0),
 datetime.datetime(1997, 7, 19, 9, 0),
 datetime.datetime(2000, 1, 1, 9, 0)]
```

Every 20th Monday of the year, 3 occurrences.

```
>>> list(rrule(YEARLY, count=3, byweekday=MO(20),
...            dtstart=parse("19970519T090000")))
[datetime.datetime(1997, 5, 19, 9, 0),
 datetime.datetime(1998, 5, 18, 9, 0),
 datetime.datetime(1999, 5, 17, 9, 0)]
```

Monday of week number 20 (where the default start of the week is Monday), 3 occurrences.

```
>>> list(rrule(YEARLY, count=3, byweekno=20, byweekday=MO,
...            dtstart=parse("19970512T090000")))
[datetime.datetime(1997, 5, 12, 9, 0),
 datetime.datetime(1998, 5, 11, 9, 0),
 datetime.datetime(1999, 5, 17, 9, 0)]
```

The week number 1 may be in the last year.

```
>>> list(rrule(WEEKLY, count=3, byweekno=1, byweekday=MO,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 12, 29, 9, 0),
 datetime.datetime(1999, 1, 4, 9, 0),
 datetime.datetime(2000, 1, 3, 9, 0)]
```

And the week numbers greater than 51 may be in the next year.

```
>>> list(rrule(WEEKLY, count=3, byweekno=52, byweekday=SU,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 12, 28, 9, 0),
 datetime.datetime(1998, 12, 27, 9, 0),
 datetime.datetime(2000, 1, 2, 9, 0)]
```

Only some years have week number 53:

```
>>> list(rrule(WEEKLY, count=3, byweekno=53, byweekday=MO,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1998, 12, 28, 9, 0),
 datetime.datetime(2004, 12, 27, 9, 0),
 datetime.datetime(2009, 12, 28, 9, 0)]
```

Every Friday the 13th, 4 occurrences.

```
>>> list(rrule(YEARLY, count=4, byweekday=FR, bymonthday=13,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1998, 2, 13, 9, 0),
 datetime.datetime(1998, 3, 13, 9, 0),
 datetime.datetime(1998, 11, 13, 9, 0),
 datetime.datetime(1999, 8, 13, 9, 0)]
```

Every four years, the first Tuesday after a Monday in November, 3 occurrences (U.S. Presidential Election day):

```
>>> list(rrule(YEARLY, interval=4, count=3, bymonth=11,
...            byweekday=TU, bymonthday=(2,3,4,5,6,7,8),
...            dtstart=parse("19961105T090000")))
[datetime.datetime(1996, 11, 5, 9, 0),
 datetime.datetime(2000, 11, 7, 9, 0),
 datetime.datetime(2004, 11, 2, 9, 0)]
```

The 3rd instance into the month of one of Tuesday, Wednesday or Thursday, for the next 3 months:

```
>>> list(rrule(MONTHLY, count=3, byweekday=(TU,WE,TH),
...            bysetpos=3, dtstart=parse("19970904T090000")))
[datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 10, 7, 9, 0),
 datetime.datetime(1997, 11, 6, 9, 0)]
```

The 2nd to last weekday of the month, 3 occurrences.

```
>>> list(rrule(MONTHLY, count=3, byweekday=(MO,TU,WE,TH,FR),
...            bysetpos=-2, dtstart=parse("19970929T090000")))
[datetime.datetime(1997, 9, 29, 9, 0),
 datetime.datetime(1997, 10, 30, 9, 0),
 datetime.datetime(1997, 11, 27, 9, 0)]
```

Every 3 hours from 9:00 AM to 5:00 PM on a specific day.

```
>>> list(rrule(HOURLY, interval=3,
...            dtstart=parse("19970902T090000"),
...            until=parse("19970902T170000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 12, 0),
 datetime.datetime(1997, 9, 2, 15, 0)]
```

Every 15 minutes for 6 occurrences.

```
>>> list(rrule(MINUTELY, interval=15, count=6,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 9, 15),
 datetime.datetime(1997, 9, 2, 9, 30),
 datetime.datetime(1997, 9, 2, 9, 45),
```

(continues on next page)

(continued from previous page)

```
 datetime.datetime(1997, 9, 2, 10, 0),
 datetime.datetime(1997, 9, 2, 10, 15)]
```

Every hour and a half for 4 occurrences.

```
>>> list(rrule(MINUTELY, interval=90, count=4,
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 10, 30),
 datetime.datetime(1997, 9, 2, 12, 0),
 datetime.datetime(1997, 9, 2, 13, 30)]
```

Every 20 minutes from 9:00 AM to 4:40 PM for two days.

```
>>> list(rrule(MINUTELY, interval=20, count=48,
...            byhour=range(9,17), byminute=(0,20,40),
...            dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 2, 9, 20),
 ...
 datetime.datetime(1997, 9, 2, 16, 20),
 datetime.datetime(1997, 9, 2, 16, 40),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 20),
 ...
 datetime.datetime(1997, 9, 3, 16, 20),
 datetime.datetime(1997, 9, 3, 16, 40)]
```

An example where the days generated makes a difference because of *wkst*.

```
>>> list(rrule(WEEKLY, interval=2, count=4,
...            byweekday=(TU,SU), wkst=MO,
...            dtstart=parse("19970805T090000")))
[datetime.datetime(1997, 8, 5, 9, 0),
 datetime.datetime(1997, 8, 10, 9, 0),
 datetime.datetime(1997, 8, 19, 9, 0),
 datetime.datetime(1997, 8, 24, 9, 0)]

>>> list(rrule(WEEKLY, interval=2, count=4,
...            byweekday=(TU,SU), wkst=SU,
...            dtstart=parse("19970805T090000")))
[datetime.datetime(1997, 8, 5, 9, 0),
 datetime.datetime(1997, 8, 17, 9, 0),
 datetime.datetime(1997, 8, 19, 9, 0),
 datetime.datetime(1997, 8, 31, 9, 0)]
```

10.7.4 rruleset examples

Daily, for 7 days, jumping Saturday and Sunday occurrences.

```
>>> set = rruleset()
>>> set.rrule(rrule(DAILY, count=7,
...                 dtstart=parse("19970902T090000")))
>>> set.exrule(rrule(YEARLY, byweekday=(SA,SU),
...                  dtstart=parse("19970902T090000")))
```

(continues on next page)

(continued from previous page)

```
>>> list(set)
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 3, 9, 0),
 datetime.datetime(1997, 9, 4, 9, 0),
 datetime.datetime(1997, 9, 5, 9, 0),
 datetime.datetime(1997, 9, 8, 9, 0)]
```

Weekly, for 4 weeks, plus one time on day 7, and not on day 16.

```
>>> set = rruleset()
>>> set.rrule(rrule(WEEKLY, count=4,
...           dtstart=parse("19970902T090000")))
>>> set.rdate(datetime.datetime(1997, 9, 7, 9, 0))
>>> set.exdate(datetime.datetime(1997, 9, 16, 9, 0))
>>> list(set)
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 7, 9, 0),
 datetime.datetime(1997, 9, 9, 9, 0),
 datetime.datetime(1997, 9, 23, 9, 0)]
```

10.7.5 rrulestr() examples

Every 10 days, 5 occurrences.

```
>>> list(rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... """))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Same thing, but passing only the *RRULE* value.

```
>>> list(rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5",
...               dtstart=parse("19970902T090000")))
[datetime.datetime(1997, 9, 2, 9, 0),
 datetime.datetime(1997, 9, 12, 9, 0),
 datetime.datetime(1997, 9, 22, 9, 0),
 datetime.datetime(1997, 10, 2, 9, 0),
 datetime.datetime(1997, 10, 12, 9, 0)]
```

Notice that when using a single rule, it returns an *rrule* instance, unless *forceset* was used.

```
>>> rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5")
<dateutil.rrule.rrule object at 0x...>

>>> rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... """)
<dateutil.rrule.rrule object at 0x...>
```

(continues on next page)

(continued from previous page)

```
>>> rrulestr("FREQ=DAILY;INTERVAL=10;COUNT=5", forceset=True)
<dateutil.rrule.rruleset object at 0x...>
```

But when an *rruleset* is needed, it is automatically used.

```
>>> rrulestr("""
... DTSTART:19970902T090000
... RRULE:FREQ=DAILY;INTERVAL=10;COUNT=5
... RRULE:FREQ=DAILY;INTERVAL=5;COUNT=3
... """)
<dateutil.rrule.rruleset object at 0x...>
```

10.8 tz

This module offers timezone implementations subclassing the abstract `datetime.tzinfo` type. There are classes to handle tzfile format files (usually are in `/etc/localtime`, `/usr/share/zoneinfo`, etc), TZ environment string (in all known formats), given ranges (with help from relative deltas), local machine timezone, fixed offset timezone, and UTC timezone.

10.8.1 Objects

`dateutil.tz.`**`UTC`**
A convenience instance of *`dateutil.tz.tzutc`*.

New in version 2.7.0.

10.8.2 Functions

`dateutil.tz.`**`gettz`**(*name=None*)
Retrieve a time zone object from a string representation

This function is intended to retrieve the `tzinfo` subclass that best represents the time zone that would be used if a POSIX TZ variable were set to the same value.

If no argument or an empty string is passed to `gettz`, local time is returned:

```
>>> gettz()
tzfile('/etc/localtime')
```

This function is also the preferred way to map IANA tz database keys to `tzfile` objects:

```
>>> gettz('Pacific/Kiritimati')
tzfile('/usr/share/zoneinfo/Pacific/Kiritimati')
```

On Windows, the standard is extended to include the Windows-specific zone names provided by the operating system:

```
>>> gettz('Egypt Standard Time')
tzwin('Egypt Standard Time')
```

Passing a GNU `TZ` style string time zone specification returns a *`tzstr`* object:

```
>>> gettz('AEST-10AEDT-11,M10.1.0/2,M4.1.0/3')
tzstr('AEST-10AEDT-11,M10.1.0/2,M4.1.0/3')
```

Parameters **name** – A time zone name (IANA, or, on Windows, Windows keys), location of a `tzfile(5)` zoneinfo file or `TZ` variable style time zone specifier. An empty string, no argument or `None` is interpreted as local time.

Returns Returns an instance of one of `dateutil`'s `tzinfo` subclasses.

Changed in version 2.7.0: After version 2.7.0, any two calls to `gettz` using the same input strings will return the same object:

```
>>> tz.gettz('America/Chicago') is tz.gettz('America/Chicago')
True
```

In addition to improving performance, this ensures that "same zone" semantics are used for datetimes in the same zone.

`gettz.`**`nocache`**`()`
A non-cached version of gettz

`classmethod` `gettz.`**`cache_clear`**`()`

`dateutil.tz.`**`enfold`**`(`*dt*, *fold=1*`)`
Provides a unified interface for assigning the `fold` attribute to datetimes both before and after the implementation of PEP-495.

Parameters **fold** – The value for the `fold` attribute in the returned datetime. This should be either 0 or 1.

Returns Returns an object for which `getattr(dt, 'fold', 0)` returns `fold` for all versions of Python. In versions prior to Python 3.6, this is a `_DatetimeWithFold` object, which is a subclass of `datetime.datetime` with the `fold` attribute added, if `fold` is 1.

New in version 2.6.0.

`dateutil.tz.`**`datetime_ambiguous`**`(`*dt*, *tz=None*`)`
Given a datetime and a time zone, determine whether or not a given datetime is ambiguous (i.e if there are two times differentiated only by their DST status).

Parameters

- **dt** – A `datetime.datetime` (whose time zone will be ignored if `tz` is provided.)
- **tz** – A `datetime.tzinfo` with support for the `fold` attribute. If `None` or not provided, the datetime's own time zone will be used.

Returns Returns a boolean value whether or not the "wall time" is ambiguous in `tz`.

New in version 2.6.0.

`dateutil.tz.`**`datetime_exists`**`(`*dt*, *tz=None*`)`
Given a datetime and a time zone, determine whether or not a given datetime would fall in a gap.

Parameters

- **dt** – A `datetime.datetime` (whose time zone will be ignored if `tz` is provided.)
- **tz** – A `datetime.tzinfo` with support for the `fold` attribute. If `None` or not provided, the datetime's own time zone will be used.

Returns Returns a boolean value whether or not the "wall time" exists in `tz`.

New in version 2.7.0.

dateutil.tz.**resolve_imaginary**(*dt*)

Given a datetime that may be imaginary, return an existing datetime.

This function assumes that an imaginary datetime represents what the wall time would be in a zone had the offset transition not occurred, so it will always fall forward by the transition's change in offset.

```
>>> from dateutil import tz
>>> from datetime import datetime
>>> NYC = tz.gettz('America/New_York')
>>> print(tz.resolve_imaginary(datetime(2017, 3, 12, 2, 30, tzinfo=NYC)))
2017-03-12 03:30:00-04:00

>>> KIR = tz.gettz('Pacific/Kiritimati')
>>> print(tz.resolve_imaginary(datetime(1995, 1, 1, 12, 30, tzinfo=KIR)))
1995-01-02 12:30:00+14:00
```

As a note, `datetime.astimezone()` is guaranteed to produce a valid, existing datetime, so a round-trip to and from UTC is sufficient to get an extant datetime, however, this generally "falls back" to an earlier time rather than falling forward to the STD side (though no guarantees are made about this behavior).

Parameters **dt** – A `datetime.datetime` which may or may not exist.

Returns Returns an existing `datetime.datetime`. If `dt` was not imaginary, the datetime returned is guaranteed to be the same object passed to the function.

New in version 2.7.0.

10.8.3 Classes

class dateutil.tz.**tzutc**

This is a tzinfo object that represents the UTC time zone.

Examples:

```
>>> from datetime import *
>>> from dateutil.tz import *

>>> datetime.now()
datetime.datetime(2003, 9, 27, 9, 40, 1, 521290)

>>> datetime.now(tzutc())
datetime.datetime(2003, 9, 27, 12, 40, 12, 156379, tzinfo=tzutc())

>>> datetime.now(tzutc()).tzname()
'UTC'
```

Changed in version 2.7.0: `tzutc()` is now a singleton, so the result of `tzutc()` will always return the same object.

```
>>> from dateutil.tz import tzutc, UTC
>>> tzutc() is tzutc()
True
>>> tzutc() is UTC
True
```

class dateutil.tz.**tzoffset**(*name*, *offset*)

A simple class for representing a fixed offset from UTC.

Parameters

- **name** – The timezone name, to be returned when `tzname()` is called.
- **offset** – The time zone offset in seconds, or (since version 2.6.0, represented as a `datetime.timedelta` object).

class dateutil.tz.**tzlocal**
A `tzinfo` subclass built around the `time` timezone functions.

class dateutil.tz.**tzwinlocal**
Class representing the local time zone information in the Windows registry

While *dateutil.tz.tzlocal* makes system calls (via the `time` module) to retrieve time zone information, `tzwinlocal` retrieves the rules directly from the Windows registry and creates an object like *dateutil.tz.tzwin*.

Because Windows does not have an equivalent of `time.tzset()`, on Windows, *dateutil.tz.tzlocal* instances will always reflect the time zone settings *at the time that the process was started*, meaning changes to the machine's time zone settings during the run of a program on Windows will **not** be reflected by *dateutil.tz.tzlocal*. Because `tzwinlocal` reads the registry directly, it is unaffected by this issue.

Note: Only available on Windows

---

display()
Return the display name of the time zone.

transitions(*year*)
For a given year, get the DST on and off transition times, expressed always on the standard time side. For zones with no transitions, this function returns `None`.

Parameters **year** – The year whose transitions you would like to query.

Returns Returns a `tuple` of `datetime.datetime` objects, `(dston, dstoff)` for zones with an annual DST transition, or `None` for fixed offset zones.

class dateutil.tz.**tzrange**(*stdabbr*, *stdoffset=None*, *dstabbr=None*, *dstoffset=None*, *start=None*, *end=None*)
The `tzrange` object is a time zone specified by a set of offsets and abbreviations, equivalent to the way the `TZ` variable can be specified in POSIX-like systems, but using Python delta objects to specify DST start, end and offsets.

Parameters

- **stdabbr** – The abbreviation for standard time (e.g. `'EST'`).
- **stdoffset** – An integer or `datetime.timedelta` object or equivalent specifying the base offset from UTC.

  If unspecified, +00:00 is used.
- **dstabbr** – The abbreviation for DST / "Summer" time (e.g. `'EDT'`).

  If specified, with no other DST information, DST is assumed to occur and the default behavior or `dstoffset`, `start` and `end` is used. If unspecified and no other DST information is specified, it is assumed that this zone has no DST.

  If this is unspecified and other DST information is *is* specified, DST occurs in the zone but the time zone abbreviation is left unchanged.

- **dstoffset** – A an integer or `datetime.timedelta` object or equivalent specifying the UTC offset during DST. If unspecified and any other DST information is specified, it is assumed to be the STD offset +1 hour.
- **start** – A `relativedelta.relativedelta` object or equivalent specifying the time and time of year that daylight savings time starts. To specify, for example, that DST starts at 2AM on the 2nd Sunday in March, pass:

  ```
  relativedelta(hours=2, month=3, day=1, weekday=SU(+2))
  ```

  If unspecified and any other DST information is specified, the default value is 2 AM on the first Sunday in April.
- **end** – A `relativedelta.relativedelta` object or equivalent representing the time and time of year that daylight savings time ends, with the same specification method as in `start`. One note is that this should point to the first time in the *standard* zone, so if a transition occurs at 2AM in the DST zone and the clocks are set back 1 hour to 1AM, set the `hours` parameter to +1.

Examples:

```
>>> tzstr('EST5EDT') == tzrange("EST", -18000, "EDT")
True

>>> from dateutil.relativedelta import *
>>> range1 = tzrange("EST", -18000, "EDT")
>>> range2 = tzrange("EST", -18000, "EDT", -14400,
...                  relativedelta(hours=+2, month=4, day=1,
...                                weekday=SU(+1)),
...                  relativedelta(hours=+1, month=10, day=31,
...                                weekday=SU(-1)))
>>> tzstr('EST5EDT') == range1 == range2
True
```

class dateutil.tz.**tzstr**(*s*, *posix_offset=False*)

`tzstr` objects are time zone objects specified by a time-zone string as it would be passed to a `TZ` variable on POSIX-style systems (see the GNU C Library: TZ Variable for more details).

There is one notable exception, which is that POSIX-style time zones use an inverted offset format, so normally `GMT+3` would be parsed as an offset 3 hours *behind* GMT. The `tzstr` time zone object will parse this as an offset 3 hours *ahead* of GMT. If you would like to maintain the POSIX behavior, pass a `True` value to `posix_offset`.

The *tzrange* object provides the same functionality, but is specified using `relativedelta.relativedelta` objects. rather than strings.

Parameters

- **s** – A time zone string in `TZ` variable format. This can be a `bytes` (2.x: `str`), `str` (2.x: `unicode`) or a stream emitting unicode characters (e.g. `StringIO`).
- **posix_offset** – Optional. If set to `True`, interpret strings such as `GMT+3` or `UTC+3` as being 3 hours *behind* UTC rather than ahead, per the POSIX standard.

Caution: Prior to version 2.7.0, this function also supported time zones in the format:

- `EST5EDT,4,0,6,7200,10,0,26,7200,3600`
- `EST5EDT,4,1,0,7200,10,-1,0,7200,3600`

This format is non-standard and has been deprecated; this function will raise a `DeprecatedTZFormatWarning` until support is removed in a future version.

class dateutil.tz.**tzical**(*fileobj*)

This object is designed to parse an iCalendar-style `VTIMEZONE` structure as set out in RFC 5545 Section 4.6.5 into one or more *tzinfo* objects.

Parameters **fileobj** – A file or stream in iCalendar format, which should be UTF-8 encoded with CRLF endings.

get(*tzid=None*)

Retrieve a `datetime.tzinfo` object by its `tzid`.

Parameters **tzid** – If there is exactly one time zone available, omitting `tzid` or passing `None` value returns it. Otherwise a valid key (which can be retrieved from *keys()*) is required.

Raises **ValueError** – Raised if `tzid` is not specified but there are either more or fewer than 1 zone defined.

Returns Returns either a `datetime.tzinfo` object representing the relevant time zone or `None` if the `tzid` was not found.

keys()

Retrieves the available time zones as a list.

class dateutil.tz.**tzwin**(*name*)

Time zone object created from the zone info in the Windows registry

These are similar to *dateutil.tz.tzrange* objects in that the time zone data is provided in the format of a single offset rule for either 0 or 2 time zone transitions per year.

Param name The name of a Windows time zone key, e.g. "Eastern Standard Time". The full list of keys can be retrieved with *tzwin.list()*.

Note: Only available on Windows

---

display()

Return the display name of the time zone.

static list()

Return a list of all time zones known to the system.

transitions(*year*)

For a given year, get the DST on and off transition times, expressed always on the standard time side. For zones with no transitions, this function returns `None`.

Parameters **year** – The year whose transitions you would like to query.

Returns Returns a `tuple` of `datetime.datetime` objects, `(dston, dstoff)` for zones with an annual DST transition, or `None` for fixed offset zones.

10.9 tz.win

This module provides an interface to the native time zone data on Windows, including `datetime.tzinfo` implementations.

Attempting to import this module on a non-Windows platform will raise an `ImportError`.

10.9.1 Classes

class dateutil.tz.win.**tzres**(*tzres_loc='tzres.dll'*)
Class for accessing `tzres.dll`, which contains timezone name related resources.

New in version 2.5.0.

load_name(*offset*)
Load a timezone name from a DLL offset (integer).

```
>>> from dateutil.tzwin import tzres
>>> tzr = tzres()
>>> print(tzr.load_name(112))
'Eastern Standard Time'
```

Parameters **offset** – A positive integer value referring to a string from the tzres dll.

Note: Offsets found in the registry are generally of the form `@tzres.dll,-114`. The offset in this case is 114, not -114.

name_from_string(*tzname_str*)
Parse strings as returned from the Windows registry into the time zone name as defined in the registry.

```
>>> from dateutil.tzwin import tzres
>>> tzr = tzres()
>>> print(tzr.name_from_string('@tzres.dll,-251'))
'Dateline Daylight Time'
>>> print(tzr.name_from_string('Eastern Standard Time'))
'Eastern Standard Time'
```

Parameters **tzname_str** – A timezone name string as returned from a Windows registry key.

Returns Returns the localized timezone string from tzres.dll if the string is of the form *@tzres.dll,-offset*, else returns the input string.

class dateutil.tz.win.**tzwin**(*name*)
Time zone object created from the zone info in the Windows registry

These are similar to *dateutil.tz.tzrange* objects in that the time zone data is provided in the format of a single offset rule for either 0 or 2 time zone transitions per year.

Param name The name of a Windows time zone key, e.g. "Eastern Standard Time". The full list of keys can be retrieved with *tzwin.list()*.

display()
Return the display name of the time zone.

static list()
Return a list of all time zones known to the system.

transitions(*year*)
For a given year, get the DST on and off transition times, expressed always on the standard time side. For zones with no transitions, this function returns `None`.

Parameters **year** – The year whose transitions you would like to query.

Returns Returns a `tuple` of `datetime.datetime` objects, `(dston, dstoff)` for zones with an annual DST transition, or `None` for fixed offset zones.

class dateutil.tz.win.**tzwinlocal**
Class representing the local time zone information in the Windows registry

While *dateutil.tz.tzlocal* makes system calls (via the `time` module) to retrieve time zone information, `tzwinlocal` retrieves the rules directly from the Windows registry and creates an object like *dateutil.tz.tzwin*.

Because Windows does not have an equivalent of `time.tzset()`, on Windows, *dateutil.tz.tzlocal* instances will always reflect the time zone settings *at the time that the process was started*, meaning changes to the machine's time zone settings during the run of a program on Windows will **not** be reflected by *dateutil.tz.tzlocal*. Because `tzwinlocal` reads the registry directly, it is unaffected by this issue.

display()
Return the display name of the time zone.

transitions(*year*)
For a given year, get the DST on and off transition times, expressed always on the standard time side. For zones with no transitions, this function returns `None`.

Parameters **year** – The year whose transitions you would like to query.

Returns Returns a `tuple` of `datetime.datetime` objects, `(dston, dstoff)` for zones with an annual DST transition, or `None` for fixed offset zones.

10.10 utils

This module offers general convenience and utility functions for dealing with datetimes.

New in version 2.7.0.

dateutil.utils.**default_tzinfo**(*dt*, *tzinfo*)
Sets the `tzinfo` parameter on naive datetimes only

This is useful for example when you are provided a datetime that may have either an implicit or explicit time zone, such as when parsing a time zone string.

```
>>> from dateutil.tz import tzoffset
>>> from dateutil.parser import parse
>>> from dateutil.utils import default_tzinfo
>>> dflt_tz = tzoffset("EST", -18000)
>>> print(default_tzinfo(parse('2014-01-01 12:30 UTC'), dflt_tz))
2014-01-01 12:30:00+00:00
>>> print(default_tzinfo(parse('2014-01-01 12:30'), dflt_tz))
2014-01-01 12:30:00-05:00
```

Parameters

- **dt** – The datetime on which to replace the time zone
- **tzinfo** – The `datetime.tzinfo` subclass instance to assign to `dt` if (and only if) it is naive.

Returns Returns an aware `datetime.datetime`.

dateutil.utils.**today**(*tzinfo=None*)
Returns a `datetime` representing the current day at midnight

Parameters **tzinfo** – The time zone to attach (also used to determine the current day).

Returns A `datetime.datetime` object representing the current day at midnight.

dateutil.utils.**within_delta**(*dt1*, *dt2*, *delta*)
Useful for comparing two datetimes that may a negilible difference to be considered equal.

10.11 zoneinfo

dateutil.zoneinfo.**get_zonefile_instance**(*new_instance=False*)
This is a convenience function which provides a `ZoneInfoFile` instance using the data provided by the `dateutil` package. By default, it caches a single instance of the ZoneInfoFile object and returns that.

Parameters **new_instance** – If `True`, a new instance of `ZoneInfoFile` is instantiated and used as the cached instance for the next call. Otherwise, new instances are created only as necessary.

Returns Returns a `ZoneInfoFile` object.

New in version 2.6.

dateutil.zoneinfo.**gettz**(*name*)
This retrieves a time zone from the local zoneinfo tarball that is packaged with dateutil.

Parameters **name** – An IANA-style time zone name, as found in the zoneinfo file.

Returns Returns a `dateutil.tz.tzfile` time zone object.

> **Warning:** It is generally inadvisable to use this function, and it is only provided for API compatibility with earlier versions. This is *not* equivalent to `dateutil.tz.gettz()`, which selects an appropriate time zone based on the inputs, favoring system zoneinfo. This is ONLY for accessing the dateutil-specific zoneinfo (which may be out of date compared to the system zoneinfo).

Deprecated since version 2.6: If you need to use a specific zoneinfofile over the system zoneinfo, instantiate a `dateutil.zoneinfo.ZoneInfoFile` object and call `dateutil.zoneinfo.ZoneInfoFile.get(name)()` instead.

Use *get_zonefile_instance()* to retrieve an instance of the dateutil-provided zoneinfo.

dateutil.zoneinfo.**gettz_db_metadata**()
Get the zonefile metadata

See *zonefile_metadata*

Returns A dictionary with the database metadata

Deprecated since version 2.6: See deprecation warning in `zoneinfo.gettz()`. To get metadata, query the attribute `zoneinfo.ZoneInfoFile.metadata`.

dateutil.zoneinfo.rebuild.**rebuild**(*filename*, *tag=None*, *format='gz'*, *zonegroups=[]*, *metadata=None*)
Rebuild the internal timezone info in dateutil/zoneinfo/zoneinfo*tar*

filename is the timezone tarball from `ftp.iana.org/tz`.

10.11.1 zonefile_metadata

The zonefile metadata defines the version and exact location of the timezone database to download. It is used in the `updatezinfo.py` script. A json encoded file is included in the source-code, and within each tar file we produce. The json file is attached here:

```
{
    "metadata_version": 2.0,
    "releases_url": [
        "https://dateutil.github.io/tzdata/tzdata/",
        "ftp://ftp.iana.org/tz/releases/"
    ],
    "tzdata_file": "tzdata2019c.tar.gz",
    "tzdata_file_sha512":
↪"2921cbb2fd44a6b8f7f2ed42c13fbae28195aa5c2eeefa70396bc97cdbaad679c6cc3c143da82cca5b0279065c02389e9a
↪",
    "tzversion": "2019c",
    "zonegroups": [
        "africa",
        "antarctica",
        "asia",
        "australasia",
        "europe",
        "northamerica",
        "southamerica",
        "pacificnew",
        "etcetera",
        "systemv",
        "factory",
        "backzone",
        "backward"
    ]
}
```

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

R

T

U

V

W