DateParser Documentation

Release 1.4.1

Scrapinghub

Jun 23, 2026

CONTENTS

CHAPTER

ONE

KEY FEATURES

- Support for almost every existing date format: absolute dates, relative dates (`"two weeks ago"` or `"tomorrow"`), timestamps, etc.
- Support for more than 200 language locales.
- Language autodetection
- Customizable behavior through settings.
- Support for *non-Gregorian calendar systems*.
- Support for dates with timezones abbreviations or UTC offsets (`"August 14, 2015 EST"`, `"21 July 2013 10:15 pm +0500"`...)
- *Search dates* in longer texts.
- Time span detection for expressions like “past month”, “last week”.

CHAPTER

TWO

ONLINE DEMO

Do you want to try it out without installing any dependency? Now you can test it quickly by visiting this online demo!

CHAPTER

THREE

HOW TO USE

The most straightforward way to parse dates with **dateparser** is to use the `dateparser.parse()` function, that wraps around most of the functionality of the module.

```
>>> import dateparser

>>> dateparser.parse('Fri, 12 Dec 2014 10:55:50')
datetime.datetime(2014, 12, 12, 10, 55, 50)

>>> dateparser.parse('1991-05-17')
datetime.datetime(1991, 5, 17, 0, 0)

>>> dateparser.parse('In two months')  # today is 1st Aug 2020
datetime.datetime(2020, 10, 1, 11, 12, 27, 764201)

>>> dateparser.parse('1484823450')  # timestamp
datetime.datetime(2017, 1, 19, 10, 57, 30)

>>> dateparser.parse('January 12, 2012 10:00 PM EST')
datetime.datetime(2012, 1, 12, 22, 0, tzinfo=<StaticTzInfo 'EST'>)
```

dateparser also works with strings in different languages:

```
>>> dateparser.parse('Martes 21 de Octubre de 2014')  # Spanish (Tuesday 21 October 2014)
datetime.datetime(2014, 10, 21, 0, 0)

>>> dateparser.parse('Le 11 Décembre 2014 à 09:00')  # French (11 December 2014 at 09:00)
datetime.datetime(2014, 12, 11, 9, 0)

>>> dateparser.parse('13  2015 .  13:34')  # Russian (13 January 2015 at 13:34)
datetime.datetime(2015, 1, 13, 13, 34)

>>> dateparser.parse('1  2005, 1:00 AM')  # Thai (1 October 2005, 1:00 AM)
datetime.datetime(2005, 10, 1, 1, 0)

>>> dateparser.parse('yaklaşık 23 saat önce')  # Turkish (23 hours ago), current time:␣
↪12:46
datetime.datetime(2019, 9, 7, 13, 46)

>>> dateparser.parse('2')  # Chinese (2 hours ago), current time: 22:30
datetime.datetime(2018, 5, 31, 20, 30)
```

You can specify the language(s), if known, using the `languages` argument. In this case, given languages are used and language detection is skipped:

```
>>> dateparser.parse('2015, Ago 15, 1:08 pm', languages=['pt', 'es'])
datetime.datetime(2015, 8, 15, 13, 8)
```

If you know the possible formats of the dates, you can use the `date_formats` argument:

```
>>> dateparser.parse('22 Décembre 2010', date_formats=['%d %B %Y'])
datetime.datetime(2010, 12, 22, 0, 0)
```

3.1 Relative Dates

```
>>> from dateparser import parse
>>> parse('1 hour ago')
datetime.datetime(2015, 5, 31, 23, 0)
>>> parse('Il ya 2 heures')  # French (2 hours ago)
datetime.datetime(2015, 5, 31, 22, 0)
>>> parse('1 anno 2 mesi')  # Italian (1 year 2 months)
datetime.datetime(2014, 4, 1, 0, 0)
>>> parse('yaklaşık 23 saat önce')  # Turkish (23 hours ago)
datetime.datetime(2015, 5, 31, 1, 0)
>>> parse('Hace una semana')  # Spanish (a week ago)
datetime.datetime(2015, 5, 25, 0, 0)
>>> parse('2')  # Chinese (2 hours ago)
datetime.datetime(2015, 5, 31, 22, 0)
```

Note: Testing above code might return different values depending on your environment's current date and time.

Note: For the `Finnish` language, please specify `settings={'SKIP_TOKENS': []}` to correctly parse relative dates.

3.2 Date Order

```
>>> # parsing ambiguous date
>>> parse('02-03-2016')  # assumes english language, uses MDY date order
datetime.datetime(2016, 2, 3, 0, 0)
>>> parse('le 02-03-2016')  # detects french, uses DMY date order
datetime.datetime(2016, 3, 2, 0, 0)
```

Note: Ordering is not locale-based — do not expect `DMY` order for UK/Australia English. You can specify date order explicitly:

```
>>> parse('18-12-15 06:00', settings={'DATE_ORDER': 'DMY'})
datetime.datetime(2015, 12, 18, 6, 0)
```

For more on date order, see the settings documentation.

3.3 Timezone and UTC Offset

By default, *dateparser* returns a timezone-aware `datetime` if a timezone is present in the date string. Otherwise it returns a naive `datetime` object.

```
>>> parse('January 12, 2012 10:00 PM EST')
datetime.datetime(2012, 1, 12, 22, 0, tzinfo=<StaticTzInfo 'EST'>)

>>> parse('January 12, 2012 10:00 PM -0500')
datetime.datetime(2012, 1, 12, 22, 0, tzinfo=<StaticTzInfo 'UTC\-05:00'>)

>>> parse('2 hours ago EST')
datetime.datetime(2017, 3, 10, 15, 55, 39, 579667, tzinfo=<StaticTzInfo 'EST'>)
```

If the date has no timezone name/abbreviation or offset, you can specify it using the `TIMEZONE` setting:

```
>>> parse('January 12, 2012 10:00 PM', settings={'TIMEZONE': 'US/Eastern'})
datetime.datetime(2012, 1, 12, 22, 0)

>>> parse('January 12, 2012 10:00 PM', settings={'TIMEZONE': 'US/Eastern', 'RETURN_AS_
↪TIMEZONE_AWARE': True})
datetime.datetime(2012, 1, 12, 22, 0, tzinfo=<DstTzInfo 'US/Eastern' EST-1 day, 19:00:00␣
↪STD>)

>>> parse('10:00 am', settings={'TIMEZONE': 'EST', 'TO_TIMEZONE': 'EDT'})
datetime.datetime(2016, 9, 25, 11, 0)

>>> parse('10:00 am EST', settings={'TO_TIMEZONE': 'EDT'})
datetime.datetime(2017, 3, 12, 11, 0, tzinfo=<StaticTzInfo 'EDT'>)
```

For more on timezones, see the settings documentation.

3.4 Incomplete Dates

```
>>> from dateparser import parse
>>> parse('December 2015')  # default behavior
datetime.datetime(2015, 12, 16, 0, 0)
>>> parse('December 2015', settings={'PREFER_DAY_OF_MONTH': 'last'})
datetime.datetime(2015, 12, 31, 0, 0)
>>> parse('December 2015', settings={'PREFER_DAY_OF_MONTH': 'first'})
datetime.datetime(2015, 12, 1, 0, 0)

>>> parse('March')
datetime.datetime(2015, 3, 16, 0, 0)
>>> parse('March', settings={'PREFER_DATES_FROM': 'future'})
datetime.datetime(2016, 3, 16, 0, 0)
```

(continues on next page)

(continued from previous page)

```
>>> import dateparser
>>> dateparser.parse("2015")  # default behavior
datetime.datetime(2015, 3, 27, 0, 0)
>>> dateparser.parse("2015", settings={"PREFER_MONTH_OF_YEAR": "last"})
datetime.datetime(2015, 12, 27, 0, 0)
>>> dateparser.parse("2015", settings={"PREFER_MONTH_OF_YEAR": "current"})
datetime.datetime(2015, 3, 27, 0, 0)
```

You can also ignore incomplete dates by setting the `STRICT_PARSING` flag:

```
>>> parse('December 2015', settings={'STRICT_PARSING': True})
None
```

For more on handling incomplete dates, see the settings documentation.

3.5 Search for Dates in Longer Chunks of Text

> **Warning:** Support for date searching is limited and needs improvement. Contributions are welcome — see contributing.

You can extract dates from longer strings of text. Results are returned as a list of `(substring, datetime)` tuples:

```
>>> from dateparser.search import search_dates
>>> search_dates('Today is 25 of October 2017, so the 27th is in 2 days.')
[('25 of October 2017', datetime.datetime(2017, 10, 25, 0, 0)), ('the 27th is in 2 days',
↪ datetime.datetime(2017, 10, 27, 0, 0))]
```

3.6 Time Span Detection

The `search_dates` function can also detect time spans such as "past month" or "last week". When `RETURN_TIME_SPAN` is enabled it returns start and end dates for the detected period:

```
>>> search_dates("Messages from the past month", settings={'RETURN_TIME_SPAN': True})
[('past month (start)', datetime.datetime(2024, 11, 7, 0, 0)),
 ('past month (end)', datetime.datetime(2024, 12, 7, 23, 59, 59, 999999))]
```

3.7 Settings

You can control multiple behaviors by using the `settings` parameter:

```
>>> dateparser.parse('2014-10-12', settings={'DATE_ORDER': 'YMD'})
datetime.datetime(2014, 10, 12, 0, 0)

>>> dateparser.parse('2014-10-12', settings={'DATE_ORDER': 'YDM'})
datetime.datetime(2014, 12, 10, 0, 0)
```

(continues on next page)

(continued from previous page)

```
>>> dateparser.parse('1 year', settings={'PREFER_DATES_FROM': 'future'})  # Today is␣
↪2020-09-23
datetime.datetime(2021, 9, 23, 0, 0)

>>> dateparser.parse('tomorrow', settings={'RELATIVE_BASE': datetime.datetime(1992, 1,␣
↪1)})
datetime.datetime(1992, 1, 2, 0, 0)
```

To see all available settings, check the settings documentation.

3.8 False positives

dateparser will do its best to return a date, dealing with multiple formats and different locales. For that reason it is important that the input is a valid date, otherwise it could return false positives.

To reduce the possibility of receiving false positives, make sure that:

- The input string is a valid date and doesn't contain any other words or numbers.
- If you know the language or languages beforehand, you add them through the `languages` or `locales` properties.

On the other hand, if you want to exclude any of the default parsers (`timestamp`, `relative-time`...) or change the order in which they are executed, you can do so through the settings PARSERS.

CHAPTER

FOUR

INSTALLATION

Dateparser supports Python 3.10+. You can install it by doing:

```
$ pip install dateparser
```

If you want to use the jalali or hijri calendar, you need to install the `calendars` extra:

```
$ pip install dateparser[calendars]
```

CHAPTER

FIVE

SUPPORTED CALENDARS

Apart from the Gregorian calendar, *dateparser* supports the *Persian Jalali calendar* and the *Hijri/Islamic calendar*. To use them, install the `calendars` extra (see *Installation*).

Example using the Persian Jalali calendar:

```
>>> from dateparser.calendars.jalali import JalaliCalendar
>>> JalaliCalendar('    ').get_date()
DateData(date_obj=datetime.datetime(2009, 3, 20, 0, 0), period='day', locale=None)
```

Example using the Hijri/Islamic calendar:

```
>>> from dateparser.calendars.hijri import HijriCalendar
>>> HijriCalendar('17-01-1437  08:30 ').get_date()
DateData(date_obj=datetime.datetime(2015, 10, 30, 20, 30), period='day', locale=None)
```

CHAPTER

SIX

DEPENDENCIES

dateparser relies on the following libraries:

- dateutil's module `relativedelta` for its freshness parser.
- convertdate to convert *Jalali* dates to *Gregorian*.
- hijridate to convert *Hijri* dates to *Gregorian*.
- tzlocal to reliably get local timezone.
- ruamel.yaml (optional) for operations on language files.

CHAPTER

SEVEN

COMMON USE CASES

dateparser can be used for a wide variety of purposes, but it stands out when it comes to:

7.1 Consuming data from different sources:

- **Scraping**: extract dates from different places with several different formats and languages
- **IoT**: consuming data coming from different sources with different date formats
- **Tooling**: consuming dates from different logs / sources
- **Format transformations**: when transforming dates coming from different files (PDF, CSV, etc.) to other formats (database, etc).

7.2 Offering natural interaction with users:

- **Tooling and CLI**: allow users to write "3 days ago" to retrieve information.
- **Search engine**: allow people to search by date in an easy / natural format.
- **Bots**: allow users to interact with a bot easily

CHAPTER

EIGHT

YOU MAY ALSO LIKE...

- price-parser - A small library for extracting price and currency from raw text strings.
- number-parser - Library to convert numbers written in the natural language to it's equivalent numeric forms.
- Scrapy - Web crawling and web scraping framework

CHAPTER

NINE

LICENSE

BSD3-Clause

CHAPTER

TEN

INDICES AND TABLES

Contents:

10.1 Installation

At the command line:

```
$ pip install dateparser
```

Or, if you don't have pip installed:

```
$ easy_install dateparser
```

If you want to install from the latest sources, you can do:

```
$ git clone https://github.com/scrapinghub/dateparser.git
$ cd dateparser
$ pip install .
```

10.2 Using DateDataParser

dateparser.parse() uses a default parser which tries to detect language every time it is called and is not the most efficient way while parsing dates from the same source.

DateDataParser provides an alternate and efficient way to control language detection behavior.

The instance of *DateDataParser* caches the found languages and will prioritize them when trying to parse the next string.

dateparser.date.DateDataParser can also be initialized with known languages:

```
>>> ddp = DateDataParser(languages=['de', 'nl'])
>>> ddp.get_date_data('vr jan 24, 2014 12:49')
DateData(date_obj=datetime.datetime(2014, 1, 24, 12, 49), period='day', locale='nl')
>>> ddp.get_date_data('18.10.14 um 22:56 Uhr')
DateData(date_obj=datetime.datetime(2014, 10, 18, 22, 56), period='day', locale='de')
>>> ddp.get_date_data('11 July 2012')
DateData(date_obj=None, period='day', locale=None)
```

10.3 Settings

dateparser's parsing behavior can be configured by supplying settings as a dictionary to *settings* argument in `dateparser.parse()` or `DateDataParser` constructor.

Note: From *dateparser 1.0.0* when a setting with a wrong value is provided, a `SettingValidationError` is raised.

All supported *settings* with their usage examples are given below:

10.3.1 Date Order

`DATE_ORDER`: specifies the order in which date components *year*, *month* and *day* are expected while parsing ambiguous dates. It defaults to `MDY` which translates to *month* first, *day* second and *year* last order. Characters *M*, *D* or *Y* can be shuffled to meet required order. For example, `DMY` specifies *day* first, *month* second and *year* last order:

```
>>> parse('15-12-18 06:00')  # assumes default order: MDY
datetime.datetime(2018, 12, 15, 6, 0)  # since 15 is not a valid value for Month, it is␣
↪swapped with Day's
>>> parse('15-12-18 06:00', settings={'DATE_ORDER': 'YMD'})
datetime.datetime(2015, 12, 18, 6, 0)
```

`PREFER_LOCALE_DATE_ORDER`: defaults to `True`. Most languages have a default `DATE_ORDER` specified for them. For example, for French it is `DMY`:

```
>>> # parsing ambiguous date
>>> parse('02-03-2016')  # assumes english language, uses MDY date order
datetime.datetime(2016, 2, 3, 0, 0)
>>> parse('le 02-03-2016')  # detects french, hence, uses DMY date order
datetime.datetime(2016, 3, 2, 0, 0)
```

Note: There's no language level default `DATE_ORDER` associated with *en* language. That's why it assumes `MDY` which is :obj:`settings <dateparser.conf.settings>` default. If the language has a default `DATE_ORDER` associated, supplying custom date order will not be applied unless we set `PREFER_LOCALE_DATE_ORDER` to `False`:

```
>>> parse('le 02-03-2016', settings={'DATE_ORDER': 'MDY'})
datetime.datetime(2016, 3, 2, 0, 0)  # MDY didn't apply
```

```
>>> parse('le 02-03-2016', settings={'DATE_ORDER': 'MDY', 'PREFER_LOCALE_DATE_ORDER':␣
↪False})
datetime.datetime(2016, 2, 3, 0, 0)  # MDY worked!
```

10.3.2 Timezone Related Configurations

`TIMEZONE`: defaults to local timezone. When specified, resultant `datetime` is localized with the given timezone. Can be timezone abbreviation or any of tz database name as given here.

```
>>> parse('January 12, 2012 10:00 PM', settings={'TIMEZONE': 'US/Eastern'})
datetime.datetime(2012, 1, 12, 22, 0)
```

`TO_TIMEZONE`: defaults to None. When specified, resultant `datetime` converts according to the supplied timezone:

```
>>> settings = {'TIMEZONE': 'UTC', 'TO_TIMEZONE': 'US/Eastern'}
>>> parse('January 12, 2012 10:00 PM', settings=settings)
datetime.datetime(2012, 1, 12, 17, 0)
```

`RETURN_AS_TIMEZONE_AWARE`: if `True` returns tz aware datetime objects in case timezone is detected in the date string.

```
>>> parse('30 mins ago', settings={'RETURN_AS_TIMEZONE_AWARE': True})
datetime.datetime(2017, 3, 13, 1, 43, 10, 243565, tzinfo=<DstTzInfo 'Asia/Karachi'␣
↪PKT+5:00:00 STD>)
```

```
>>> parse('12 Feb 2015 10:56 PM EST', settings={'RETURN_AS_TIMEZONE_AWARE': False})
datetime.datetime(2015, 2, 12, 22, 56)
```

10.3.3 Handling Incomplete Dates

`PREFER_DAY_OF_MONTH`: it comes handy when the date string is missing the day part. It defaults to `current` and can be `first` and `last` denoting first and last day of months respectively as values:

```
>>> from dateparser import parse
>>> parse('December 2015')  # default behavior
datetime.datetime(2015, 12, 16, 0, 0)
>>> parse('December 2015', settings={'PREFER_DAY_OF_MONTH': 'last'})
datetime.datetime(2015, 12, 31, 0, 0)
>>> parse('December 2015', settings={'PREFER_DAY_OF_MONTH': 'first'})
datetime.datetime(2015, 12, 1, 0, 0)
```

`PREFER_MONTH_OF_YEAR`: Similarly, another useful thing when the date string is missing the month part. It defaults to `current` and can be `first` and `last` denoting first and last month of year respectively as values:

```
>>> from dateparser import parse
>>> parse("2015") # default behavior
datetime.datetime(2015, 3, 27, 0, 0)
>>> parse("2015", settings={"PREFER_MONTH_OF_YEAR": "last"})
datetime.datetime(2015, 12, 27, 0, 0)
>>> parse("2015", settings={"PREFER_MONTH_OF_YEAR": "first"})
datetime.datetime(2015, 1, 27, 0, 0)
>>> parse("2015", settings={"PREFER_MONTH_OF_YEAR": "current"}) # it exactly behaves␣
↪like default one
datetime.datetime(2015, 3, 27, 0, 0)
```

`PREFER_DATES_FROM`: defaults to `current_period` and can have `past` and `future` as values.

If date string is missing some part, this option ensures consistent results depending on the `past` or `future` preference, for example, assuming current date is *June 16, 2015*:

```
>>> from dateparser import parse
>>> parse('March')
datetime.datetime(2015, 3, 16, 0, 0)
>>> parse('March', settings={'PREFER_DATES_FROM': 'future'})
datetime.datetime(2016, 3, 16, 0, 0)
>>> # parsing with preference set for 'past'
>>> parse('August', settings={'PREFER_DATES_FROM': 'past'})
datetime.datetime(2015, 8, 15, 0, 0)
```

`RELATIVE_BASE`: allows setting the base datetime to use for interpreting partial or relative date strings. Defaults to the current date and time.

For example, assuming current date is *June 16, 2015*:

```
>>> from dateparser import parse
>>> parse('14:30')
datetime.datetime(2015, 6, 16, 14, 30)
>>> parse('14:30', settings={'RELATIVE_BASE': datetime.datetime(2020, 1, 1)})
datetime.datetime(2020, 1, 1, 14, 30)
>>> parse('tomorrow', settings={'RELATIVE_BASE': datetime.datetime(2020, 1, 1)})
datetime.datetime(2020, 1, 2, 0, 0)
```

RETURN_TIME_SPAN

default: `False`

When enabled, *search_dates* detects time span expressions (e.g., "past month", "last week") and returns start/end dates instead of single dates.

DEFAULT_START_OF_WEEK

default: `'monday'`

Sets which day starts the week for time span calculations. Options: `'monday'`, `'sunday'`.

DEFAULT_DAYS_IN_MONTH

default: `30`

Sets the number of days to use for "past month" and similar expressions in time span detection.

`STRICT_PARSING`: defaults to `False`.

When set to `True` if missing any of `day`, `month` or `year` parts, it does not return any result altogether.:

```
>>> parse('March', settings={'STRICT_PARSING': True})
None
```

`REQUIRE_PARTS`: ensures results are dates that have all specified parts. It defaults to `[]` and can include `day`, `month` and/or `year`.

For example, assuming current date is *June 16, 2019*:

```
>>> parse('2012') # default behavior
datetime.datetime(2012, 6, 16, 0, 0)
>>> parse('2012', settings={'REQUIRE_PARTS': ['month']})
None
>>> parse('March 2012', settings={'REQUIRE_PARTS': ['day']})
None
>>> parse('March 12, 2012', settings={'REQUIRE_PARTS': ['day']})
datetime.datetime(2012, 3, 12, 0, 0)
>>> parse('March 12, 2012', settings={'REQUIRE_PARTS': ['day', 'month', 'year']})
datetime.datetime(2012, 3, 12, 0, 0)
```

10.3.4 Language Detection

SKIP_TOKENS: it is a `list` of tokens to discard while detecting language. Defaults to `['t']` which skips T in iso format datetime string .e.g. `2015-05-02T10:20:19+0000`.:

```
>>> from dateparser.date import DateDataParser
>>> DateDataParser(settings={'SKIP_TOKENS': ['de']}).get_date_data(u'27 Haziran 1981 de
↪') # Turkish (at 27 June 1981)
DateData(date_obj=datetime.datetime(1981, 6, 27, 0, 0), period='day', locale='tr')
```

NORMALIZE: applies unicode normalization (removing accents, diacritics...) when parsing the words. Defaults to True.

```
>>> dateparser.parse('4 decembre 2015', settings={'NORMALIZE': False})
# It doesn't work as the expected input should be '4 décembre 2015'
```

```
>>> dateparser.parse('4 decembre 2015', settings={'NORMALIZE': True})
datetime.datetime(2015, 12, 4, 0, 0)
```

10.3.5 Default Languages

DEFAULT_LANGUAGES: It is a `list` of language codes in ISO 639 that will be used as default languages for parsing when language detection fails. eg. ["en", "fr"]:

```
>>> from dateparser import parse
>>> parse('3 de marzo de 2020', settings={'DEFAULT_LANGUAGES': ["es"]})
```

Note: When using this setting, these languages will be tried after trying with the detected languages with no success. It is especially useful when using `detect_languages_function`.

10.3.6 Language Order

USE_GIVEN_LANGUAGE_ORDER: defaults to False. By default, the languages and locales passed through the languages and locales arguments are tried in the order of the most common languages, regardless of the order in which they are given. Set this to True to instead try them in the order in which they are given, which is useful when that order already reflects a preference (for example, the output of a language detector ordered by confidence):

```
>>> import dateparser
>>> dateparser.parse('11/12/2020', languages=['es', 'en'])
datetime.datetime(2020, 11, 12, 0, 0)
>>> dateparser.parse('11/12/2020', languages=['es', 'en'], settings={'USE_GIVEN_LANGUAGE_
↪ORDER': True})
datetime.datetime(2020, 12, 11, 0, 0)
```

It also applies to the languages given through the DEFAULT_LANGUAGES setting when they are used as a fallback.

Note: This setting only reorders the languages and locales that are given. If neither languages nor locales is provided, it has no effect and the default order is used:

```
>>> dateparser.parse('11/12/2020', settings={'USE_GIVEN_LANGUAGE_ORDER': True})
datetime.datetime(2020, 11, 12, 0, 0)
```

Note: This setting does not affect dateparser.search.search_dates(), which detects a single most likely language and normalizes the text before parsing, so it does not use the given language order to disambiguate values such as 11/12.

10.3.7 Optional language detection

LANGUAGE_DETECTION_CONFIDENCE_THRESHOLD: defaults to 0.5. It is a float of minimum required confidence for the custom language detection:

```
>>> from dateparser import parse
>>> parse('3 de marzo de 2020', settings={'LANGUAGE_DETECTION_CONFIDENCE_THRESHOLD': 0.5}
↪, detect_languages_function=detect_languages)
```

10.3.8 Other settings

RETURN_TIME_AS_PERIOD: returns time as period in date object, if time component is present in date string. Defaults to False.

```
>>> ddp = DateDataParser(settings={'RETURN_TIME_AS_PERIOD': True})
>>> ddp.get_date_data('vr jan 24, 2014 12:49')
DateData(date_obj=datetime.datetime(2014, 1, 24, 12, 49), period='time', locale='nl')
```

PARSERS: it is a list of names of parsers to try, allowing to customize which parsers are tried against the input date string, and in which order they are tried.

The following parsers exist:

- `'timestamp'`: If the input string starts with 10 digits, optionally followed by additional digits or a period (`.`), those first 10 digits are interpreted as Unix time.
- `'negative-timestamp'`: `'timestamp'` for negative timestamps. For example, parses `-186454800000` as `1964-02-03T23:00:00`.
- `'relative-time'`: Parses dates and times expressed in relation to the current date and time (e.g. "1 day ago", "in 2 weeks").
- `'custom-formats'`: Parses dates that match one of the date formats in the list of the `date_formats` parameter of *dateparser.parse()* or *DateDataParser.get_date_data*.
- `'absolute-time'`: Parses dates and times expressed in absolute form (e.g. "May 4th", "1991-05-17"). It takes into account settings such as `DATE_ORDER` or `PREFER_LOCALE_DATE_ORDER`.
- `'no-spaces-time'`: Parses dates and times that consist in only digits or a combination of digits and non-digits where the first non-digit it's a colon (e.g. "121994", "11:052020"). It's not included in the default parsers and it can produce false positives frequently.

`dateparser.settings.default_parsers` contains the default value of `PARSERS` (the list above, in that order) and can be used to write code that changes the parsers to try without skipping parsers that may be added to Dateparser in the future. For example, to ignore relative times:

```
>>> from dateparser_data.settings import default_parsers
>>> parsers = [parser for parser in default_parsers if parser != 'relative-time']
>>> parse('today', settings={'PARSERS': parsers})
```

`CACHE_SIZE_LIMIT`: limits the size of caches, that store data for already processed dates. Default to `1000`, but you can set `0` for turning off the limit.

10.4 Custom language detection

dateparser allows to customize the language detection behavior by using the `detect_languages_function` parameter. It supports the langdetect library out of the box, and allows you to implement your own custom language detection.

Warning: For short strings the language detection could fail, so it's highly recommended to use `detect_languages_function` along with `DEFAULT_LANGUAGES`.

10.4.1 Built-in implementations

langdetect

Language detection with langdetect.

To use langdetect, first install it:

```
pip install dateparser[langdetect]
```

Then import the langdetect wrapper and pass it as `detect_languages_function` parameter. Example:

```
>>> from dateparser.custom_language_detection.langdetect import detect_languages
>>> dateparser.parse('12/12/12', detect_languages_function=detect_languages)
```

Note: **Deprecated fastText support**: The fastText integration has been removed as the library is archived and no longer maintained. If you were using fastText, please migrate to langdetect.

10.4.2 Custom implementation

`dateparser` allows the integration of any library to detect languages by wrapping that library in a function that accepts 2 parameters, `text` and `confidence_threshold`, and returns a list of the detected language codes in ISO 639 standards.

Wrapper for boilerplate for implementing custom language detections:

```
def detect_languages(text, confidence_threshold):
    """
    Takes 2 parameters, `text` and `confidence_threshold`, and returns
    a list of `languages codes`.

    * `text` is the input string whose language needs to be detected.

    * `confidence_threshold` is a number between 0 and 1 that indicates the
    minimum confidence required for language matches.

    For language detection libraries that, for each language, indicate how
    confident they are that the language matches the input text, you should
    filter out languages with a confidence lower than this value (adjusted,
    if needed, to the confidence range of the target library).

    This value comes from the dateparser setting
    `LANGUAGE_DETECTION_CONFIDENCE_THRESHOLD`.

    The result must be a list of languages codes (strings).
    """
    # here you can apply your own logic
    return language_codes
```

10.5 Supported languages and locales

The `region` argument accepts the region subtag from one of the locale codes listed below. For example, `languages=['en'], region='IN'` uses the `en-IN` locale, while `languages=['fr'], region='CA'` uses `fr-CA`. If you already know the full locale code, pass it directly through `locales=['en-IN']` instead.

Language	Locales
af	'af-NA'
agq	
ak	
am	
ar	'ar-AE', 'ar-BH', 'ar-DJ', 'ar-DZ', 'ar-EG', 'ar-EH', 'ar-ER', 'ar-IL', 'ar-IQ', 'ar-JO', 'ar-KM', 'ar-KW', 'ar-LB', 'ar-LY',
as	
asa	

Language	Locales
ast	
az	
az-Cyrl	
az-Latn	
bas	
be	
bem	
bez	
bg	
bm	
bn	‘bn-IN’
bo	‘bo-IN’
br	
brx	
bs	
bs-Cyrl	
bs-Latn	
ca	‘ca-AD’, ‘ca-FR’, ‘ca-IT’
ce	
cgg	
chr	
ckb	‘ckb-IR’
cs	
cy	
da	‘da-GL’
dav	
de	‘de-AT’, ‘de-BE’, ‘de-CH’, ‘de-IT’, ‘de-LI’, ‘de-LU’
dje	
dsb	
dua	
dyo	
dz	
ebu	
ee	‘ee-TG’
el	‘el-CY’
en	‘en-001’, ‘en-150’, ‘en-AG’, ‘en-AI’, ‘en-AS’, ‘en-AT’, ‘en-AU’, ‘en-BB’, ‘en-BE’, ‘en-BI’, ‘en-BM’, ‘en-BS’, ‘en-BW’,
eo	
es	‘es-419’, ‘es-AR’, ‘es-BO’, ‘es-BR’, ‘es-BZ’, ‘es-CL’, ‘es-CO’, ‘es-CR’, ‘es-CU’, ‘es-DO’, ‘es-EA’, ‘es-EC’, ‘es-GQ’, ‘es
et	
eu	
ewo	
fa	‘fa-AF’
ff	‘ff-CM’, ‘ff-GN’, ‘ff-MR’
fi	
fil	
fo	‘fo-DK’
fr	‘fr-BE’, ‘fr-BF’, ‘fr-BI’, ‘fr-BJ’, ‘fr-BL’, ‘fr-CA’, ‘fr-CD’, ‘fr-CF’, ‘fr-CG’, ‘fr-CH’, ‘fr-CI’, ‘fr-CM’, ‘fr-DJ’, ‘fr-DZ’, ‘fr-G
fur	
fy	
ga	

Language	Locales
gd	
gl	
gsw	‘gsw-FR’, ‘gsw-LI’
gu	
guz	
gv	
ha	‘ha-GH’, ‘ha-NE’
haw	
he	
hi	
hr	‘hr-BA’
hsb	
hu	
hy	
id	
ig	
ii	
is	
it	‘it-CH’, ‘it-SM’, ‘it-VA’
ja	
jgo	
jmc	
ka	
kab	
kam	
kde	
kea	
khq	
ki	
kk	
kl	
kln	
km	
kn	
ko	‘ko-KP’
kok	
ks	
ksb	
ksf	
ksh	
kw	
ky	
lag	
lb	
lg	
lkt	
ln	‘ln-AO’, ‘ln-CF’, ‘ln-CG’
lo	
lrc	‘lrc-IQ’
lt	

Language	Locales
lu	
luo	
luy	
lv	
mas	‘mas-TZ’
mer	
mfe	
mg	
mgh	
mgo	
mk	
ml	
mn	
mr	
ms	‘ms-BN’, ‘ms-SG’
mt	
mua	
my	
mzn	
naq	
nb	‘nb-SJ’
nd	
ne	‘ne-IN’
nl	‘nl-AW’, ‘nl-BE’, ‘nl-BQ’, ‘nl-CW’, ‘nl-SR’, ‘nl-SX’
nmg	
nn	
nnh	
nus	
nyn	
om	‘om-KE’
or	
os	‘os-RU’
pa	
pa-Arab	
pa-Guru	
pl	
ps	
pt	‘pt-AO’, ‘pt-CH’, ‘pt-CV’, ‘pt-GQ’, ‘pt-GW’, ‘pt-LU’, ‘pt-MO’, ‘pt-MZ’, ‘pt-PT’, ‘pt-ST’, ‘pt-TL’
qu	‘qu-BO’, ‘qu-EC’
rm	
rn	
ro	‘ro-MD’
rof	
ru	‘ru-BY’, ‘ru-KG’, ‘ru-KZ’, ‘ru-MD’, ‘ru-UA’
rw	
rwk	
sah	
saq	
sbp	
se	‘se-FI’, ‘se-SE’

Language	Locales
seh	
ses	
sg	
shi	
shi-Latn	
shi-Tfng	
si	
sk	
sl	
smn	
sn	
so	‘so-DJ’, ‘so-ET’, ‘so-KE’
sq	‘sq-MK’, ‘sq-XK’
sr	
sr-Cyrl	‘sr-Cyrl-BA’, ‘sr-Cyrl-ME’, ‘sr-Cyrl-XK’
sr-Latn	‘sr-Latn-BA’, ‘sr-Latn-ME’, ‘sr-Latn-XK’
sv	‘sv-AX’, ‘sv-FI’
sw	‘sw-CD’, ‘sw-KE’, ‘sw-UG’
ta	‘ta-LK’, ‘ta-MY’, ‘ta-SG’
te	
teo	‘teo-KE’
th	
ti	‘ti-ER’
tl	
to	
tr	‘tr-CY’
twq	
tzm	
ug	
uk	
ur	‘ur-IN’
uz	
uz-Arab	
uz-Cyrl	
uz-Latn	
vi	
vun	
wae	
xog	
yav	
yi	
yo	‘yo-BJ’
yue	
zgh	
zh	
zh-Hans	‘zh-Hans-HK’, ‘zh-Hans-MO’, ‘zh-Hans-SG’
zh-Hant	‘zh-Hant-HK’, ‘zh-Hant-MO’
zu	

10.6 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

10.6.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/scrapinghub/dateparser/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs and Implement Features

Look through the GitHub issues for bugs and feature requests. To avoid duplicate efforts, try to choose issues without related PRs or with staled PRs. We also encourage you to add new languages to the existing stack.

Write Documentation

Dateparser could always use more documentation, whether as part of the official Dateparser docs, in docstrings, or even on the web in blog posts, articles, and such.

After you make local changes to the documentation, you will be able to build the project running:

```
tox -e docs
```

Then open `.tox/docs/tmp/html/index.html` in a web browser to see your local build of the documentation.

Note: If you don't have `tox` installed, you need to install it first using `pip install tox`.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/scrapinghub/dateparser/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that contributions are welcome :)

10.6.2 Get Started!

Ready to contribute? Here's how to set up *dateparser* for local development.

1. Fork the *dateparser* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dateparser.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv dateparser
$ cd dateparser/
$ pip install -e .
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
 $ tox

To get ``tox``, just ``pip install`` it into your virtualenv. In addition to tests,␣
↪``tox`` checks for code style and maximum line length (119 characters).
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

10.6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in *README.rst*.
3. Check the pipelines (Github Actions) in the PR comments (or in https://github.com/scrapinghub/dateparser/actions) and make sure that the tests pass for all supported Python versions.
4. Check the new project coverage in the PR comments (or in https://app.codecov.io/gh/scrapinghub/dateparser/pulls) and make sure that it remained equal or higher than previously.
5. Follow the core developers' advice which aims to ensure code's consistency regardless of the variety of approaches used by many contributors.
6. In case you are unable to continue working on a PR, please leave a short comment to notify us. We will be pleased to make any changes required to get it done.

10.6.4 Guidelines for Editing Translation Data

English is the primary language of Dateparser. Dates in all other languages are translated into English equivalents before they are parsed.

The language data that Dateparser uses to parse dates is in `dateparser/data/date_translation_data`. For each supported language, there is a Python file containing translation data.

Each translation data Python files contains different kinds of translation data for date parsing: month and week names - and their abbreviations, prepositions, conjunctions, frequently used descriptive words and phrases (like "today"), etc.

Translation data Python files are generated from the following sources:

- Unicode CLDR data in JSON format, located at `dateparser_data/cldr_language_data/date_translation_data`
- Additional data from the Dateparser community in YAML format, located at `dateparser_data/supplementary_language_data/date_translation_data`

If you wish to extend the data of an existing language, or add data for a new language, you must:

1. Edit or create the corresponding file within `dateparser_data/supplementary_language_data/date_translation_data`

   See existing files to learn how they are defined, and see *Language Data Template* for details.

2. Regenerate the corresponding file within `dateparser/data/date_translation_data` running the following script:

   ```
   dateparser_scripts/write_complete_data.py
   ```

3. Write tests that cover your changes

   You should be able to find tests that cover the affected data, and use copy-and-paste to create the corresponding new test.

   If in doubt, ask Dateparser maintainers for help.

Language Data Template

```
two-letter language code as defined in ISO-639-1 (https://en.wikipedia.org/wiki/List_of_
↪ISO_639-1_codes). e.g. for English - en:
    name: language name (e.g. English)
    no_word_spacing: False (set to True for languages that do not use spaces between␣
↪words)

    skip: ["words", "to", "skip", "such", "as", "and", "or", "at", "in", "alphabetical",
↪"order"]

    pertain: []

    monday:
        - name for Monday
        - abbreviation for Monday
    tuesday:
        - as above
    wednesday:
        - as above
```

(continues on next page)

(continued from previous page)

```
    thursday:
        - as above
    friday:
        - as above
    saturday:
        - as above
    sunday:
        - as above

    january:
        - name for January
        - abbreviation for January
    february:
        - as above
    march:
        - as above
    april:
        - as above
    may:
        - as above
    june:
        - as above
    july:
        - as above
    august:
        - as above
    september:
        - as above
    october:
        - as above
    november:
        - as above
    december:
        - as above

    year:
        - name for year
        - abbreviation for year
    month:
        - as above
    week:
        - as above
    day:
        - as above
    hour:
        - as above
    minute:
        - as above
    second:
        - as above

    ago:
```

(continues on next page)

(continued from previous page)

```
        - words that stand
        - for "ago"

    simplifications:
        - word: replacement
        - regex: replacement
        - day before yesterday: 2 days ago
```

10.6.5 Updating the List of Supported Languages and Locales

Whenever the content of `dateparser.data.languages_info.language_locale_dict` is modified, use `dateparser_scripts/update_supported_languages_and_locales.py` to update the corresponding documentation table:

```
dateparser_scripts/update_supported_languages_and_locales.py
```

10.7 API reference

10.7.1 dateparser package

Subpackages

dateparser.languages package

Submodules

dateparser.languages.dictionary module

class dateparser.languages.dictionary.**Dictionary**(*locale_info*, *settings=None*)

Bases: object

Class that modifies and stores translations and handles splitting of date string.

Parameters

- **locale_info** – Locale info (translation data) of the locale.
- **settings** (*dict*) – Configure customized behavior using settings defined in *dateparser.conf.Settings*.

Returns

a Dictionary instance.

are_tokens_valid(*tokens*)

Check if tokens are valid tokens for the locale.

Parameters

tokens (*list*) – a list of string tokens.

Returns

True if tokens are valid, False otherwise.

split(*string*, *keep_formatting=False*)

Split the date string using translations in locale info.

Parameters

- **string** (*str*) – Date string to be splitted.
- **keep_formatting** (*bool*) – If True, retain formatting of the date string.

Returns
A list of string tokens formed after splitting the date string.

class dateparser.languages.dictionary.**NormalizedDictionary**(*locale_info*, *settings=None*)

Bases: *Dictionary*

exception dateparser.languages.dictionary.**UnknownTokenError**

Bases: Exception

dateparser.languages.loader module

class dateparser.languages.loader.**LocaleDataLoader**

Bases: object

Class that handles loading of locale instances.

get_locale(*shortname*)

Get a locale instance.

Parameters
shortname (*str*) – A locale code, e.g. 'fr-PF', 'qu-EC', 'af-NA'.

Returns
locale instance

get_locale_map(*languages=None*, *locales=None*, *region=None*, *use_given_order=False*, *allow_conflicting_locales=False*)

Get an ordered mapping with locale codes as keys and corresponding locale instances as values.

Parameters

- **languages** (*list*) – A list of language codes, e.g. ['en', 'es', 'zh-Hant']. If locales are not given, languages and region are used to construct locales to load.
- **locales** (*list*) – A list of codes of locales which are to be loaded, e.g. ['fr-PF', 'qu-EC', 'af-NA']
- **region** (*str*) – A region code, e.g. 'IN', '001', 'NE'. If locales are not given, languages and region are used to construct locales to load.
- **use_given_order** (*bool*) – If True, the returned mapping is ordered in the order locales are given.
- **allow_conflicting_locales** (*bool*) – if True, locales with same language and different region can be loaded.

Returns
ordered locale code to locale instance mapping

get_locales(*languages=None*, *locales=None*, *region=None*, *use_given_order=False*, *allow_conflicting_locales=False*)

Yield locale instances.

Parameters

- **languages** (*list*) – A list of language codes, e.g. ['en', 'es', 'zh-Hant']. If locales are not given, languages and region are used to construct locales to load.
- **locales** (*list*) – A list of codes of locales which are to be loaded, e.g. ['fr-PF', 'qu-EC', 'af-NA']
- **region** (*str*) – A region code, e.g. 'IN', '001', 'NE'. If locales are not given, languages and region are used to construct locales to load.
- **use_given_order** (*bool*) – If True, the returned mapping is ordered in the order locales are given.
- **allow_conflicting_locales** (*bool*) – if True, locales with same language and different region can be loaded.

Yield

locale instances

dateparser.languages.locale module

class dateparser.languages.locale.**Locale**(*shortname*, *language_info*)

Bases: object

Class that deals with applicability and translation from a locale.

Parameters

- **shortname** (*str*) – A locale code, e.g. 'fr-PF', 'qu-EC', 'af-NA'.
- **language_info** (*dict*) – Language info (translation data) of the language the locale belongs to.

Returns

A Locale instance

static clean_dictionary(*dictionary*, *threshold=2*)

count_applicability(*text*, *strip_timezone=False*, *settings=None*)

get_wordchars_for_detection(*settings*)

is_applicable(*date_string*, *strip_timezone=False*, *settings=None*)

Check if the locale is applicable to translate date string.

Parameters

- **date_string** (*str*) – A string representing date and/or time in a recognizably valid format.
- **strip_timezone** (*bool*) – If True, timezone is stripped from date string.

Returns

boolean value representing if the locale is applicable for the date string or not.

to_parserinfo(*base_cls=<class 'dateutil.parser._parser.parserinfo'>*)

translate(*date_string*, *keep_formatting=False*, *settings=None*)

Translate the date string to its English equivalent.

Parameters

- **date_string** (*str*) – A string representing date and/or time in a recognizably valid format.
- **keep_formatting** (*bool*) – If True, retain formatting of the date string after translation.

Returns
translated date string.

translate_search(*search_string*, *settings=None*)

dateparser.languages.validation module

class dateparser.languages.validation.**LanguageValidator**

Bases: object

VALID_KEYS = ['name', 'skip', 'pertain', 'simplifications', 'no_word_spacing', 'ago', 'in', 'monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday', 'january', 'february', 'march', 'april', 'may', 'june', 'july', 'august', 'september', 'october', 'november', 'december', 'year', 'month', 'week', 'day', 'hour', 'minute', 'second', 'sentence_splitter_group']

classmethod get_logger()

logger = None

classmethod validate_info(*language_id*, *info*)

Module contents

dateparser.calendars package

Submodules

Module contents

class dateparser.calendars.**CalendarBase**(*source*)

Bases: object

Base setup class for non-Gregorian calendar system.

Parameters
source (*str*) – Date string passed to calendar parser.

Submodules

dateparser.conf module

exception dateparser.conf.**SettingValidationError**

Bases: ValueError

class dateparser.conf.**Settings**(**args*, ***kwargs*)

Bases: object

Control and configure default parsing behavior of dateparser. Currently, supported settings are:

- *DATE_ORDER*
- *PREFER_LOCALE_DATE_ORDER*
- *TIMEZONE*
- *TO_TIMEZONE*
- *RETURN_AS_TIMEZONE_AWARE*
- *PREFER_MONTH_OF_YEAR*
- *PREFER_DAY_OF_MONTH*
- *PREFER_DATES_FROM*
- *RELATIVE_BASE*
- *STRICT_PARSING*
- *REQUIRE_PARTS*
- *SKIP_TOKENS*
- *NORMALIZE*
- *RETURN_TIME_AS_PERIOD*
- *RETURN_TIME_SPAN*
- *DEFAULT_START_OF_WEEK*
- *DEFAULT_DAYS_IN_MONTH*
- *PARSERS*
- *DEFAULT_LANGUAGES*
- *USE_GIVEN_LANGUAGE_ORDER*
- *LANGUAGE_DETECTION_CONFIDENCE_THRESHOLD*
- *CACHE_SIZE_LIMIT*

classmethod get_key(*settings=None*)

replace(*mod_settings=None*, ***kwds*)

dateparser.conf.**apply_settings**(*f*)

dateparser.conf.**check_settings**(*settings*)

Check if provided settings are valid, if not it raises *SettingValidationError*. Only checks for the modified settings.

dateparser.date module

class dateparser.date.**DateData**(*, *date_obj=None*, *period=None*, *locale=None*)

Bases: object

Class that represents the parsed data with useful information. It can be accessed with square brackets like a dict object.

class dateparser.date.**DateDataParser**(*languages=None*, *locales=None*, *region=None*, *try_previous_locales=False*, *use_given_order=False*, *settings=None*, *detect_languages_function=None*)

Bases: object

Class which handles language detection, translation and subsequent generic parsing of string representing date and/or time.

Parameters

- **languages** (*list*) – A list of language codes, e.g. ['en', 'es', 'zh-Hant']. If locales are not given, languages and region are used to construct locales for translation.
- **locales** (*list*) – A list of locale codes, e.g. ['fr-PF', 'qu-EC', 'af-NA']. The parser uses only these locales to translate date string.
- **region** (*str*) – A region code, e.g. 'IN', '001', 'NE'. If locales are not given, languages and region are used to construct locales for translation.
- **try_previous_locales** (*bool*) – If True, locales previously used to translate date are tried first.
- **use_given_order** (*bool*) – If True, locales are tried for translation of date string in the order in which they are given. This is equivalent to the `USE_GIVEN_LANGUAGE_ORDER` setting; the given order is preserved if either is enabled.
- **settings** (*dict*) – Configure customized behavior using settings defined in *dateparser.conf.Settings*.
- **detect_languages_function** (*function*) – A function for language detection that takes as input a *text* and a *confidence_threshold*, and returns a list of detected language codes. Note: this function is only used if `languages` and `locales` are not provided.

Returns

A parser instance

Raises

`ValueError`: Unknown Language, `TypeError`: Languages argument must be a list, `SettingValidationError`: A provided setting is not valid.

get_date_data(*date_string*, *date_formats=None*)

Parse string representing date and/or time in recognizable localized formats. Supports parsing multiple languages and timezones.

Parameters

- **date_string** (*str*) – A string representing date and/or time in a recognizably valid format.
- **date_formats** (*list*) – A list of format strings using directives as given here. The parser applies formats one by one, taking into account the detected languages.

Returns

a `DateData` object.

Raises
ValueError - Unknown Language

Note: *Period* values can be a 'day' (default), 'week', 'month', 'year', 'time'.

Period represents the granularity of date parsed from the given string.

In the example below, since no day information is present, the day is assumed to be current day 16 from *current date* (which is June 16, 2015, at the moment of writing this). Hence, the level of precision is `month`:

```
>>> DateDataParser().get_date_data('March 2015')
DateData(date_obj=datetime.datetime(2015, 3, 16, 0, 0), period='month', locale=
↪'en')
```

Similarly, for date strings with no day and month information present, level of precision is `year` and day 16 and month 6 are from *current_date*.

```
>>> DateDataParser().get_date_data('2014')
DateData(date_obj=datetime.datetime(2014, 6, 16, 0, 0), period='year', locale=
↪'en')
```

Dates with time zone indications or UTC offsets are returned in UTC time unless specified using Settings.

```
>>> DateDataParser().get_date_data('23 March 2000, 1:21 PM CET')
DateData(date_obj=datetime.datetime(2000, 3, 23, 13, 21, tzinfo=<StaticTzInfo
↪'CET'>),
period='day', locale='en')
```

get_date_tuple(**args*, ***kwargs*)

locale_loader = None

dateparser.date.**date_range**(*begin*, *end*, ***kwargs*)

dateparser.date.**get_date_from_timestamp**(*date_string*, *settings*, *negative=False*)

dateparser.date.**get_intersecting_periods**(*low*, *high*, *period='day'*)

dateparser.date.**parse_with_formats**(*date_string*, *date_formats*, *settings*)
Parse with formats and return a dictionary with 'period' and 'obj_date'.

Returns
datetime.datetime, dict or None

dateparser.date.**sanitize_date**(*date_string*)

dateparser.date.**sanitize_spaces**(*date_string*)

dateparser.date_parser module

class dateparser.date_parser.**DateParser**

Bases: object

parse(*date_string*, *parse_method*, *settings=None*)

dateparser.freshness_date_parser module

class dateparser.freshness_date_parser.**FreshnessDateDataParser**

Bases: object

Parses date string like "1 year, 2 months ago" and "3 hours, 50 minutes ago"

get_date_data(*date_string*, *settings=None*)

get_kwargs(*date_string*)

get_local_tz()

parse(*date_string*, *settings*)

dateparser.timezone_parser module

class dateparser.timezone_parser.**StaticTzInfo**(*name*, *offset*)

Bases: tzinfo

dst(*dt*)

datetime -> DST offset as timedelta positive east of UTC.

localize(*dt*, *is_dst=False*)

tzname(*dt*)

datetime -> string name of time zone.

utcoffset(*dt*)

datetime -> timedelta showing offset from UTC, negative values indicating West of UTC

dateparser.timezone_parser.**build_tz_offsets**(*search_regex_parts*)

dateparser.timezone_parser.**convert_to_local_tz**(*datetime_obj*, *datetime_tz_offset*)

dateparser.timezone_parser.**get_local_tz_offset**()

dateparser.timezone_parser.**pop_tz_offset_from_string**(*date_string*, *as_offset=True*)

dateparser.timezone_parser.**word_is_tz**(*word*)

dateparser.timezones module

dateparser.utils module

dateparser.utils.**apply_dateparser_timezone**(*utc_datetime*, *offset_or_timezone_abb*)

dateparser.utils.**apply_timezone**(*date_time*, *tz_string*)

dateparser.utils.**apply_timezone_from_settings**(*date_obj*, *settings*)

dateparser.utils.**apply_tzdatabase_timezone**(*date_time*, *pytz_string*)

dateparser.utils.**combine_dicts**(*primary_dict*, *supplementary_dict*)

dateparser.utils.**find_date_separator**(*format*)

dateparser.utils.**get_last_day_of_month**(*year*, *month*)

dateparser.utils.**get_logger**()

dateparser.utils.**get_next_leap_year**(*year*)

dateparser.utils.**get_previous_leap_year**(*year*)

dateparser.utils.**get_timezone_from_tz_string**(*tz_string*)

dateparser.utils.**localize_timezone**(*date_time*, *tz_string*)

dateparser.utils.**normalize_unicode**(*string*, *form='NFKD'*)

dateparser.utils.**registry**(*cls*)

dateparser.utils.**set_correct_day_from_settings**(*date_obj*, *settings*, *current_day=None*)
Set correct day attending the *PREFER_DAY_OF_MONTH* setting.

dateparser.utils.**set_correct_month_from_settings**(*date_obj*, *settings*, *current_month=None*)
Set correct month attending the *PREFER_MONTH_OF_YEAR* setting.

dateparser.utils.**setup_logging**()

dateparser.utils.**strip_braces**(*date_string*)

Module contents

dateparser.**parse**(*date_string*, *date_formats=None*, *languages=None*, *locales=None*, *region=None*, *settings=None*, *detect_languages_function=None*)

Parse date and time from given date string.

Parameters

- **date_string** (*str*) – A string representing date and/or time in a recognizably valid format.
- **date_formats** (*list*) – A list of format strings using directives as given here. The parser applies formats one by one, taking into account the detected languages/locales.
- **languages** (*list*) – A list of language codes, e.g. [‘en’, ‘es’, ‘zh-Hant’]. If locales are not given, languages and region are used to construct locales for translation.
- **locales** (*list*) – A list of locale codes, e.g. [‘fr-PF’, ‘qu-EC’, ‘af-NA’]. The parser uses only these locales to translate date string.

- **region** (*str*) – A region code, e.g. 'IN', '001', 'NE'. If locales are not given, languages and region are used to construct locales for translation.
- **settings** (*dict*) – Configure customized behavior using settings defined in *dateparser.conf.Settings*.
- **detect_languages_function** (*function*) – A function for language detection that takes as input a string (the *date_string*) and a *confidence_threshold*, and returns a list of detected language codes. Note: this function is only used if `languages` and `locales` are not provided.

Returns
Returns `datetime` representing parsed date if successful, else returns None

Return type
`datetime`.

Raises
`ValueError`: Unknown Language, `TypeError`: Languages argument must be a list, `SettingValidationError`: A provided setting is not valid.

10.8 Credits

Currently, more than 100 committers have contributed to this project, making this contributors list really hard to maintain, so we have decided to stop updating this list.

To see the people behind this code, you can run `git shortlog -s -n` or visit the contributions section in Github: https://github.com/scrapinghub/dateparser/graphs/contributors

We really appreciate **all the people that has contributed to this project with their time and ideas**. Special mention to **Waqas Shabir** (waqasshabbir), **Eugene Amirov** (Allactaga) and **Artur Sadurski** (asadurski) for creating and maintaining this awesome project.

To all of you… thank you for building and improving this!

10.9 History

10.9.1 1.4.1 (2026-06-15)

Breaking changes:

- Remove fastText language detection support: the `fasttext` extra is dropped and `detect_languages()` now raises `ImportError`. Migrate to the `langdetect` extra, which also unblocks `numpy` 2.x compatibility (#1315)

Security fixes:

- Make digit quantifiers possessive in the relative-date regexes to prevent quadratic backtracking (ReDoS) on long digit runs (#1335)

New features:

- Add the `USE_GIVEN_LANGUAGE_ORDER` setting to try `languages` and `locales` in the order given rather than by frequency (#789)

Fixes:

- Preserve explicit signs on individual components when parsing relative dates that combine decades with years, such as "-1 decade +2 years" (#1330)

- Fall back to other provided languages in `search_dates` when the detected language yields no dates (#1331)
- Parse relative date expressions with spaces between the sign and number, such as "now - 2 hours" and "now + 1 day" (#1327)
- Use the parser-relative `now` for the current month when filling in incomplete dates so the month and day stay consistent (#1332)
- Fix Norwegian Bokmål (`nb`) parsing of relative date expressions such as "3 måneder siden" and "om 2 måneder" (#1334)
- Parse abbreviated English month expressions such as "1mon ago" and "3mons ago" (#1329)
- Preserve surrounding whitespace when removing skip tokens during translation to avoid spurious double spaces (#1324)

Improvements:

- Move project metadata and build configuration to `pyproject.toml` (#1311)
- Add alternative Korean date expressions for today, yesterday, tomorrow, and "N months ago/later" (#1289)
- Expand Czech date translations with additional inflections, word numbers, decade and century expressions, and clock phrases like "čtvrt na tři" (#1325)
- Replace internal `OrderedDict` usage with the built-in `dict` (#1328)

10.9.2 1.4.0 (2026-03-26)

Security fixes:

- Remove import-time loading of timezone offset data from pickle to prevent unsafe deserialization from packaged data
- Replace `eval()` use when parsing `no_word_spacing` with strict boolean parsing to prevent code execution from locale metadata (#1056)

New features:

- Add support for expressions like "N {interval} from now" in English (#1271)
- Add support for the `en-US` locale (#1222)

Fixes:

- Honor `REQUIRE_PARTS` for ambiguous month-number inputs by retrying with a year-biased `DATE_ORDER` (#1298)
- Fix parsing word-number relative phrases such as "two days later" (#1316)
- Allow md5hash to work in FIPS environments (#1267)

Improvements:

- Add Bosnian Cyrillic (ijekavica) date translations (#1293)
- Add a new browser-based demo to the project documentation (#1306)
- Update installation documentation to replace `setup.py install` guidance (#1310)
- Add a project security policy (#1318)

10.9.3 1.3.0 (2026-02-04)

Dropped Python 3.9 support. (#1296)

New features:

- `search_dates()` can now detect time spans from expressions like "past month", "last week", etc. For details, see the "Time Span Detection" section and the `RETURN_TIME_SPAN`, `DEFAULT_START_OF_WEEK` and `DEFAULT_DAYS_IN_MONTH` settings in the documentation. (#1284)

Fixes:

- Assume the current year if not specified (#1288)
- Support expressions like "yesterday +1h" (#1303)
- English: Support most 2-letter day-of-the-week names (#1214)
- English: Support "in N weeks' time" (#1283)
- Finnish: Support dates with "klo" (#1301)
- Russian: Support compound ordinals (#1280)

Cleanups and internal improvements:

- Fixed year expectation issues in tests. (#1294)

10.9.4 1.2.2 (2025-06-26)

Fixes:

- Handle the Russian preposition "" (#1261)
- Fix weekday search (#1274)

Improvements:

- Add Python 3.14 support (#1273)
- Cache timezone offsets to improve import time (#1250)

10.9.5 1.2.1 (2025-02-05)

Fixes:

- Fix PytzUsageWarning (#1109)
- Fix date_parser with prefer_month_of_year wrong results (#1224)
- Fix skipped day when UTC and tz are different days (#1183)

Improvements:

- Avoid repeated loop over timezones (#1238)
- Proofread README.rst (#1234)
- Check for derived types for configuration (#1223)
- Parse some abbreviated strings as relative dates (#1219)
- Migrate from hijri-converter to hijridate (#1211)
- Fixed ClusterFuzz build error by adding dateparser.data as a binary (#1208)

- Fix an issue detected by OSSFuzz (#1203)
- Support two-digit years in non-Gregorian calendars (#1187)
- Refactored CI to run extras separately and test minimum versions of dependencies, replaced flake8 with ruff, fixed tests (#1248)
- Set minimum versions for dependencies (#1248)
- Limited `numpy` to 1.x when installing `dateparser[fasttext]` (#1248)

10.9.6 1.2.0 (2023-11-17)

New features:

- New `PREFER_MONTH_OF_YEAR` setting (#1146)

Fixes:

- Absolute years in Russian are no longer being treated as a number of years in the past (#1129)

Cleanups and internal improvements:

- Removed the use of `datetime.utcnow`, deprecated on Python 3.12 (#1179)
- Applied Black formatting to the code base (#1158)
- Initial integration with OSSFuzz (#1198)
- Extended test cases (#1191)

10.9.7 1.1.8 (2023-03-22)

Improvements:

- Improved date parsing for Chinese (#1148)
- Improved date parsing for Czech (#1151)
- Reorder language by popularity (#1152)
- Fix leak of memory in cache (#1140)
- Add support for “d units later” (#1154)
- Move modification in CLDR data to yaml (#1153)
- Add support to use timezone via settings to get PREFER_DATES_FROM result (#1155)

10.9.8 1.1.7 (2023-02-02)

Improvements:

- Add an “ago” synonym for Arabic (#1128)
- Improved date parsing for Czech (#1131)
- Improved date parsing for Indonesian (#1134)

10.9.9 1.1.6 (2023-01-12)

Improvements:

- Fix the bug where Monday is parsed as a month (#1121)
- Prevent ReDoS in Spanish sentence splitting regex (#1084)

10.9.10 1.1.5 (2022-12-29)

Improvements:

- Parse short versions of day, month, and year (#1103)
- Add a test for "in 1d" (#1104)
- Update languages_info (#1107)
- Add a workaround for zipimporter not having exec_module before Python 3.10 (#1069)
- Stabilize tests at midnight (#1111)
- Add a test case for French (#1110)

Cleanups:

- Remove the requirements-build file (#1113)

10.9.11 1.1.4 (2022-11-21)

Improvements:

- Improved support for languages such as Slovak, Indonesian, Hindi, German and Japanese (#1064, #1094, #986, #1071, #1068)
- Recursively create a model home (#996)
- Replace regex sub with simple string replace (#1095)
- Add Python 3.10, 3.11 support (#1096)
- Drop support for Python 3.5, 3.6 versions (#1097)

10.9.12 1.1.3 (2022-11-03)

New features:

- Add support for fractional units (#876)

Improvements:

- Fix the returned datetime skipping a day with time+timezone input and PREFER_DATES_FROM = 'future' (#1002)
- Fix input translatation breaking keep_formatting (#720)
- English: support "till date" (#1005)
- English: support "after" and "before" in relative dates (#1008)

Cleanups:

- Reorganize internal data (#1090)

- CI updates (#1088)

10.9.13 1.1.2 (2022-10-20)

Improvements:

- Added support for negative timestamp (#1060)
- Fixed PytzUsageWarning for Python versions >= 3.6 (#1062)
- Added support for dates with dots and spaces (#1028)
- Improved support for Ukrainian, Croatian and Russian (#1072, #1074, #1079, #1082, #1073, #1083)
- Added support for parsing Unix timestamps consistently regardless of timezones (#954)
- Improved tests (#1086)

10.9.14 1.1.1 (2022-03-17)

Improvements:

- Fixed issue with regex library by pinning dependencies to an earlier version (< 2022.3.15, #1046).
- Extended support for Russian language dates starting with lowercase (#999).
- Allowed to use_given_order for languages too (#997).
- Fixed link to settings section (#1018).
- Defined UTF-8 encoding for Windows (#998).
- Fixed directories creation error in CLI utils (#1022).

10.9.15 1.1.0 (2021-10-04)

New features:

- Support language detection based on `langdetect`, `fastText`, or a custom implementation (see #932)
- Add support for 'by <time>' (see #839)
- Sort default language list by internet usage (see #805)

Improvements:

- Improved support of Chinese (#910), Czech (#977)
- Improvements in `search_dates` (see #953)
- Make order of previous locales deterministic (see #851)
- Fix parsing with trailing space (see #841)
- Consider `RETURN_TIME_AS_PERIOD` for timestamp times (see #922)
- Exclude failing regex version (see #974)
- Ongoing work multithreading support (see #881, #885)
- Add demo URL (see #883)

QA:

- Migrate pipelines from Travis CI to Github Actions (see #859, #879, #884, #886, #911, #966)
- Use versioned CLDR data (see #825)
- Add a script to update table of supported languages and locales (see #601)
- Sort 'skip' keys in yaml files (see #844)
- Improve test coverage (see #827)
- Code cleanup (see #888, #907, #951, #958, #957)

10.9.16 1.0.0 (2020-10-29)

Breaking changes:

- Drop support for Python 2.7 and pypy (see #727, #744, #748, #749, #754, #755, #758, #761, #763, #764, #777 and #783)
- Now `DateDataParser.get_date_data()` returns a `DateData` object instead of a `dict` (see #778).
- From now wrong `settings` are not silenced and raise `SettingValidationError` (see #797)
- Now `dateparser.parse()` is deterministic and doesn't try previous locales. Also, `DateDataParser.get_date_data()` doesn't try the previous locales by default (see #781)
- Remove the `'base-formats'` parser (see #721)
- Extract the `'no-spaces-time'` parser from the `'absolute-time'` parser and make it an optional parser (see #786)
- Remove `numeral_translation_data` (see #782)
- Remove the undocumented `SKIP_TOKENS_PARSER` and `FUZZY` settings (see #728, #794)
- Remove support for using strings in `date_formats` (see #726)
- The undocumented `ExactLanguageSearch` class has been moved to the private scope and some internal methods have changed (see #778)
- Changes in `dateparser.utils`: `normalize_unicode()` doesn't accept `bytes` as input and `convert_to_unicode` has been deprecated (see #749)

New features:

- Add Python 3.9 support (see #732, #823)
- Detect hours separated with a period/dot (see #741)
- Add support for "decade" (see #762)
- Add support for the hijri calendar in Python 3.6 (see #718)

Improvements:

- New logo! (see #719)
- Improve the README and docs (see #779, #722)
- Fix the "calendars" extra (see #740)
- Fix leap years when `PREFER_DATES_FROM` is set (see #738)
- Fix `STRICT_PARSING` setting in `no-spaces-time` parser (see #715)
- Consider `RETURN_AS_TIME_PERIOD` setting for `relative-time` parser (see #807)
- Parse the 24hr time format with meridian info (see #634)

- Other small improvements (see #698, #709, #710, #712, #730, #731, #735, #739, #784, #788, #795 and #801)

10.9.17 0.7.6 (2020-06-12)

Improvements:

- Rename `scripts` to `dateparser_scripts` to avoid name collisions with modules from other packages or projects (see #707)

10.9.18 0.7.5 (2020-06-10)

New features:

- Add Python 3.8 support (see #664)
- Implement a `REQUIRE_PARTS` setting (see #703)
- Add support for subscript and superscript numbers (see #684)
- Extended French support (see #672)
- Extended German support (see #673)

Improvements:

- Migrate test suite to Pytest (see #662)
- Add test to check the *yaml* and *json* files content (see #663 and #692)
- Add flake8 pipeline with pytest-flake8 (see #665)
- Add partial support for 8-digit dates without separators (see #639)
- Fix possible `OverflowError` errors and explicitly avoid to raise `ValueError` when parsing relative dates (see #686)
- Fix double-digit GMT and UTC parsing (see #632)
- Fix bug when using `DATE_ORDER` (see #628)
- Fix bug when parsing relative time with timezone (see #503)
- Fix milliseconds parsing (see #572 and #661)
- Fix wrong values to be interpreted as `'future'` in `PREFER_DATES_FROM` (see #629)
- Other small improvements (see #667, #675, #511, #626, #512, #509, #696, #702 and #699)

10.9.19 0.7.4 (2020-03-06)

New features:

- Extended Norwegian support (see #598)
- Implement a `PARSERS` setting (see #603)

Improvements:

- Add support for `PREFER_DATES_FROM` in relative/freshness parser (see #414)
- Add support for `PREFER_DAY_OF_MONTH` in base-formats parser (see #611)
- Added UTC -00:00 as a valid offset (see #574)

- Fix support for “one” (see #593)
- Fix TypeError when parsing some invalid dates (see #536)
- Fix tokenizer for non recognized characters (see #622)
- Prevent installing regex 2019.02.19 (see #600)
- Resolve DeprecationWarning related to raw string escape sequences (see #596)
- Implement a tox environment to build the documentation (see #604)
- Improve tests stability (see #591, #605)
- Documentation improvements (see #510, #578, #619, #614, #620)
- Performance improvements (see #570, #569, #625)

10.9.20 0.7.3 (2020-03-06)

- Broken version

10.9.21 0.7.2 (2019-09-17)

Features:

- Extended Czech support
- Added `time` to valid periods
- Added timezone information to dates found with `search_dates()`
- Support strings as date formats

Improvements:

- Fixed Collections ABCs depreciation warning
- Fixed dates with trailing colons not being parsed
- Fixed date format override on any settings change
- Fixed parsing current weekday as past date, regardless of settings
- Added UTC -2:30 as a valid offset
- Added Python 3.7 to supported versions, dropped support for Python 3.3 and 3.4
- Moved to importlib from imp where possible
- Improved support for Catalan
- Documentation improvements

10.9.22 0.7.1 (2019-02-12)

Features/news:

- Added detected language to return value of `search_dates()`
- Performance improvements
- Refreshed versions of dependencies

Improvements:

- Fixed unpickleable `DateTime` objects with timezones
- Fixed regex pattern to avoid new behaviour of re.split in Python 3.7
- Fixed an exception thrown when parsing colons
- Fixed tests failing on days with number greater than 30
- Fixed `ZeroDivisionError` exceptions

10.9.23 0.7.0 (2018-02-08)

Features added during Google Summer of Code 2017:

- Harvesting language data from Unicode CLDR database (https://github.com/unicode-cldr/cldr-json), which includes over 200 locales (#321) - authored by Sarthak Maddan. See full currently supported locale list in README.
- Extracting dates from longer strings of text (#324) - authored by Elena Zakharova. Special thanks for their awesome contributions!

New features:

- Added (independently from CLDR) Georgian (#308) and Swedish (#305)

Improvements:

- Improved support of Chinese (#359), Thai (#345), French (#301, #304), Russian (#302)
- Removed ruamel.yaml from dependencies (#374). This should reduce the number of installation issues and improve performance as the result of moving away from YAML as basic data storage format. Note that YAML is still used as format for support language files.
- Improved performance through using pre-compiling frequent regexes and lazy loading of data (#293, #294, #295, #315)
- Extended tests (#316, #317, #318, #323)
- Updated nose_parameterized to its current package, parameterized (#381)

Planned for next release:

- Full language and locale names
- Performance and stability improvements
- Documentation improvements

10.9.24 0.6.0 (2017-03-13)

New features:

- Consistent parsing in terms of true python representation of date string. See #281
- Added support for Bangla, Bulgarian and Hindi languages.

Improvements:

- Major bug fixes related to parser and system's locale. See #277, #282
- Type check for timezone arguments in settings. see #267
- Pinned dependencies' versions in requirements. See #265
- Improved support for cn, es, dutch languages. See #274, #272, #285

Packaging:

- Make calendars extras to be used at the time of installation if need to use calendars feature.

10.9.25 0.5.1 (2016-12-18)

New features:

- Added support for Hebrew

Improvements:

- Safer loading of YAML. See #251
- Better timezone parsing for freshness dates. See #256
- Pinned dependencies' versions in requirements. See #265
- Improved support for zh, fi languages. See #249, #250, #248, #244

10.9.26 0.5.0 (2016-09-26)

New features:

- `DateDataParser` now also returns detected language in the result dictionary.
- Explicit and lucid timezone conversion for a given datestring using `TIMEZONE`, `TO_TIMEZONE` settings.
- Added Hungarian language.
- Added setting, `STRICT_PARSING` to ignore incomplete dates.

Improvements:

- Fixed quite a few parser bugs reported in issues #219, #222, #207, #224.
- Improved support for chinese language.
- Consistent interface for both Jalali and Hijri parsers.

10.9.27 0.4.0 (2016-06-17)

New features:

- Support for Language based date order preference while parsing ambiguous dates.
- Support for parsing dates with no spaces in between components.
- Support for custom date order preference using `settings`.
- Support for parsing generic relative dates in future.e.g. "tomorrow", "in two weeks", etc.
- Added `RELATIVE_BASE` settings to set date context to any datetime in past or future.
- Replaced `dateutil.parser.parse` with dateparser's own parser.

Improvements:

- Added simplifications for "12 noon" and "12 midnight".
- Fixed several bugs
- Replaced PyYAML library by its active fork *ruamel.yaml* which also fixed the issues with installation on windows using python35.
- More predictable `date_formats` handling.

10.9.28 0.3.5 (2016-04-27)

New features:

- Danish language support.
- Japanese language support.
- Support for parsing date strings with accents.

Improvements:

- Transformed languages.yaml into base file and separate files for each language.
- Fixed vietnamese language simplifications.
- No more version restrictions for python-dateutil.
- Timezone parsing improvements.
- Fixed test environments.
- Cleaned language codes. Now we strictly follow codes as in ISO 639-1.
- Improved chinese dates parsing.

10.9.29 0.3.4 (2016-03-03)

Improvements:

- Fixed broken version 0.3.3 by excluding latest python-dateutil version.

10.9.30 0.3.3 (2016-02-29)

New features:

- Finnish language support.

Improvements:

- Faster parsing with switching to regex module.
- `RETURN_AS_TIMEZONE_AWARE` setting to return tz aware date object.
- Fixed conflicts with month/weekday names similarity across languages.

10.9.31 0.3.2 (2016-01-25)

New features:

- Added Hijri Calendar support.
- Added settings for better control over parsing dates.
- Support to convert parsed time to the given timezone for both complete and relative dates.

Improvements:

- Fixed problem with caching `datetime.now()` in `FreshnessDateDataParser`.
- Added month names and week day names abbreviations to several languages.
- More simplifications for Russian and Ukrainian languages.
- Fixed problem with parsing time component of date strings with several kinds of apostrophes.

10.9.32 0.3.1 (2015-10-28)

New features:

- Support for Jalali Calendar.
- Belarusian language support.
- Indonesian language support.

Improvements:

- Extended support for Russian and Polish.
- Fixed bug with time zone recognition.
- Fixed bug with incorrect translation of “second” for Portuguese.

10.9.33 0.3.0 (2015-07-29)

New features:

- Compatibility with Python 3 and PyPy.

Improvements:

- *languages.yaml* data cleaned up to make it human-readable.
- Improved Spanish date parsing.

10.9.34 0.2.1 (2015-07-13)

- Support for generic parsing of dates with UTC offset.
- Support for Tagalog/Filipino dates.
- Improved support for French and Spanish dates.

10.9.35 0.2.0 (2015-06-17)

- Easy to use `parse` function
- Languages definitions using YAML.
- Using translation based approach for parsing non-english languages. Previously, `dateutil.parserinfo` was used for language definitions.
- Better period extraction.
- Improved tests.
- Added a number of new simplifications for more comprehensive generic parsing.
- Improved validation for dates.
- Support for Polish, Thai and Arabic dates.
- Support for `pytz` timezones.
- Fixed building and packaging issues.

10.9.36 0.1.0 (2014-11-24)

- First release on PyPI.
- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

U

V

W