qddate is a Python 3 lib that helps to parse any date strings from html pages extremely fast. This lib was created during long term news aggregation efforts and analyzing in wild HTML pages with dates. It’s not intended to have beautiful code, support for so much languages as possible and so on. It should help to process millions of strings to identify and parse dates. qddata was part of proprietary technology of “news reconstruction”. It’s used to automatically create RSS feeds from sites without it.

If you are looking for more advanced (and slower) date parsing try dateparser and dateutil.
Documentation is built automatically and can be found on Read the Docs.
Features

• More than 348 date patterns supported (by the end 2017)
• Generic parsing of dates in English, Russian, Spanish, Portuguese and other languages
• Supports strings with left aligned dates and supplemental words. Example: “12.03.1999 some text here”
• Extremely fast, uses pyparsing, hard-coded constants and dirty speed optimizations tricks
CHAPTER 3

Limitations

• Not all languages supported, more languages will be added by request and example
• Not so easy to add new language based date patterns as it’s in dateparser for example.
• Could miss some rarely used date formats
• Doesn’t support relative dates
• Doesn’t support calendars
• All constants are hard encoded, no external settings
• Uses only datetime and pyparsing as external libraries. No more dependencies, all reused code incorporated into the lib code
• No regular expressions, instead pre-generated pyparsing patterns
• Intensive pattern filtering using min/max text length filters and common text patterns
• No one settings/data file loaded from disk
Usage

The easiest way is to use the `qddate.DateParser` class, and it’s `parse` function.

```python
class qddate.DateParser(generate=True)
    Class to use pyparsing-based patterns to parse dates

    match(text, noprefix=False)
        Matches date/datetime string against date patterns and returns pattern and parsed date if matched. It’s not
        indeded for common usage, since if successful it returns date as array of numbers and pattern that matched
        this date

        Parameters
            • text – Any human readable string
            • noprefix (bool) – If set True than doesn’t use prefix based date patterns filtering settings

        Returns
            Returns dicts with values as array of representing parsed date and ‘pattern’ with info about matched pattern if successful, else returns None

        Return type dict.

date = parse(text, noprefix=False)
    Parse date and time from given date string.

    Parameters
            • text – Any human readable string
            • noprefix (bool) – If set True than doesn’t use prefix based date patterns filtering settings

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

    Return type datetime.
```
This will try to parse a date from the given string, attempting to detect the language each time.
 Dependencies

$qddate$ relies on following libraries in some ways:

- $pyparsing$ is a module for advanced text processing.
CHAPTER 7

Supported languages

- Bulgarian
- Czech
- English
- French
- German
- Portuguese
- Russian
- Spanish
CHAPTER 8

Thanks

I wrote this date parsing code at 2008 year and later only updated it several times, migrating from regular expressions to pyparsing. Looking at dateparser <https://github.com/scrapinghub/dateparser> clean code and documentation motivated me to return to this code and to clean it up and to share it publicly. I’ve used same documentation and code style approach and reused build scripts and documentation generation style from dateutil. Many thanks to ScrapingHub team!

8.1 Using DateParser.match

DateParser is the only way to implement fast dates parsing.

The instance of DateParser uses basic date patterns from qddateconsts and generates extended list of patterns. It helps to reduce number of comparisons of strings significantly. Right now no language selection implemented but it doesn’t slow down date parsing.

This class wraps around the core qddate functionality.

class qddate.DateParser (generate=True)
Class to use pyparsing-based patterns to parse dates

match (text, noprefix=False)
Matches date/datetime string against date patterns and returns pattern and parsed date if matched. It’s not indeded for common usage, since if successful it returns date as array of numbers and pattern that matched this date

Parameters

• text – Any human readable string

• noprefix (bool) – If set True than doesn’t use prefix based date patterns filtering settings

Returns Returns dicts with values as array of representing parsed date and 'pattern' with info about matched pattern if successful, else returns None

Return type dict.
Warning: It returns raw matched date and raw pattern:

```
>>> dp.match('11 August 2017')
{'values': ([11], 8, ['2017']), 'pattern': {'key': 'dt:date:date_eng1', 'name': 'Date with english month', 'pattern': {'W': (0123...) Suppress:('[.']) January | February | March | April | May | June | July | August | September | October | November | December Suppress:('[.']) W: (0123...), 'length': {'min': 10, 'max': 20}, 'format': '%d.%b.%Y', 'right': True, 'basekey': 'dt:date:date_eng1'))
```

## 8.2 Popular Formats

```python
class qddate.DateParser(generate=True)
    Class to use pyparsing-based patterns to parse dates
    parse(text, noprefix=False)
        Parse date and time from given date string.

Parameters

- text -- Any human readable string
- noprefix (bool) -- If set True than doesn’t use prefix based date patterns filtering settings

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

Return type datetime.
```

Function ‘parse’ mimics default behavior of dateparser ‘parse’ function. Except that it is part of DateParser class, not standalone function.

```python
>>> import qddate
>>> parser = qddate.DateParser()
>>> parser.parse('2012-12-15')
datetime.datetime(2012, 12, 15, 0, 0)
>>> parser.parse('Fri, 12 Dec 2014 10:55:50')
datetime.datetime(2014, 12, 12, 10, 55, 50)
>>> parser.parse('17, 2015')  # Russian (17 July 2015)
datetime.datetime(2015, 1, 13, 13, 34)
>>> dp.parse('Le 8 juillet 2015')
datetime.datetime(2015, 7, 8, 0, 0)
```

This will try to parse a date from the given string, attempting to detect the language each time automatically.
9.1 Installation

At the command line:

$ pip install qddate

Or, if you don’t have pip installed:

$ easy_install qddate

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

$ git clone https://github.com/ivbeg/qddate.git
$ cd qddate
$ python setup.py qddate

9.1.1 Dependencies

Similarly, you can download source and package pyparsing as eggs and deploy them like above.

9.2 Benchmark

Benchmark code qddate vs dateparser is at ‘tests/bench.py’. It runs 10 times against 50 raw dates, so it’s up to 500 comparisons.

Latest run: >>> Bench per 10 pass: qddate 0.5 seconds, dateparser 44.8 seconds >>> qddate.DateParser.parse is 84.7X faster over dateparser.parse
Note: This benchmarking was for internal testing only. Independent benchmark will be highly appreciated. Feel free to write me at ivan@begtn.tech

9.3 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:

9.3.1 Types of Contributions

Report Bugs

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

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it. We encourage you to add new languages to 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.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ivbeg/qddate/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 :)

9.3.2 Get Started!

Ready to contribute? Here’s how to set up qddate for local development.

1. Fork the qddate repo on GitHub.
2. Clone your fork locally:
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 qddate
   $ cd qddate/
   $ python setup.py develop

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:

   $ pip install -r tests/requirements.txt # install test dependencies
   $ flake8 qddate tests
   $ nosetests
   $ tox

   To get flake8 and tox, just pip install them into your virtualenv. (Note that we use max-line-length = 100 for flake8, this is configured in setup.cfg file.)
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.

9.3.3 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 https://travis-ci.org/ivbeg/qddate/pull_requests and make sure that the tests pass for all supported Python versions.
4. Follow the core developers’ advice which aim to ensure code’s consistency regardless of variety of approaches used by many contributors.
5. 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.
9.3.4 Guidelines for Adding New Languages

English and Russian are the primary languages of the qddate. All languages are pre-configured and manually added for speed optimization. DateParser has function \_matchPrefix which filters patterns and minimizes date comparisons.

Languages added using DATE_DATA_TYPES_RAW constant in consts.py. It’s combined, inherited patterns and it's manually configured and not so simplified yet.

9.4 Credits

9.4.1 Committers

• Ivan Begtin

9.5 History

9.5.1 0.1.1 (2018-07-20)

• Code cleanup, date patterns moved to “qddate.patterns”

9.5.2 0.1.0 (2018-01-14)

• First public release on PyPI and github
CHAPTER 10

Indices and tables

- genindex
- modindex
- search
Index

D
DateParser (class in qddate), 11, 17, 18

M
match() (qddate.DateParser method), 11, 17

P
parse() (qddate.DateParser method), 11, 18

Q
qddate (module), 11