Python time expressions parser Documentation

Release 0.1.0

Ouahib El Hanchi

Apr 11, 2018

Contents:

CHAPTER 1

Python time expressions parser

Basic Python module to parse time expressions.

- Free software: MIT license
- Documentation: https://pytimeparser.readthedocs.io.

1.1 Usage

To use Python time expressions parser in a project:

```
import pytimeparser

output_timedelta = pytimeparser.parse('3 days 5:12:43.123')

print(output_timedelta.total_seconds())
```

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Python time expressions parser, run this command in your terminal:

```
$ pip install pytimeparser
```

This is the preferred method to install Python time expressions parser, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Python time expressions parser can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ouahibelhanchi/pytimeparser
```

Or download the tarball:

```
$ curl  -OL https://github.com/ouahibelhanchi/pytimeparser/tarball/master
```

Once you have a copy of the source, you can install it with:

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

CHAPTER 3

Usage

To use Python time expressions parser in a project:

```
import pytimeparser

pytimeparser.parse('3 days 5:12:43.123')
```

CHAPTER 4

pytimeparser

4.1 pytimeparser package

4.1.1 Submodules

4.1.2 pytimeparser.api module

pytimeparser.api.**parse**(*text*)
Parse time expression.

Parameters **text** – Time expression to parse

Returns datetime.timedelta object

Return type datetime.timedelta

Raises TypeError: if text is not a string

Raises ValueError: if text is empty

4.1.3 pytimeparser.utils module

Utility functions

pytimeparser.utils.**expand_regex**(*regex*, *text*, *template*, *sep=' '*)
Expand regex using provided template

pytimeparser.utils.**mapping_inversed**(*mapping*)
Inverse a mapping of lists

pytimeparser.utils.**mapping_to_list**(*mapping*)
Convert a mapping to a list

pytimeparser.utils.**normalize_numbers**(*text*)
Normalize numbers.

pytimeparser.utils.**normalize_text**(*text*)
Normalize text by normalizing whitespace and numbers.

pytimeparser.utils.**normalize_whitespace**(*text*, *replace_with=' '*)
Normalize whitespace.

pytimeparser.utils.**remove_whitespace**(*text*)
Remove all whitespace.

4.1.4 Module contents

Top-level package for Python time expressions parser.

pytimeparser.**parse**(*text*)
Parse time expression.

Parameters **text** – Time expression to parse

Returns `datetime.timedelta` object

Return type datetime.timedelta

Raises TypeError: if text is not a string

Raises ValueError: if text is empty

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/ouahibelhanchi/pytimeparser/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ouahibelhanchi/pytimeparser/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 this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/pytimeparser.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 pytimeparser
$ cd pytimeparser/
$ 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:

```
$ flake8 pytimeparser tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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.

5.3 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. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/ouahibelhanchi/pytimeparser/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_pytimeparser
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Ouahib El Hanchi <ouahib.el.hanchi@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2018-04-02)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index