reagex

Release 0.1.2

Dec 16, 2018

Contents

CHAPTER 1

Overview

docs	
tests	
package	

The goal of `reagex` (from *"readable regular expression"*) is to suggest a way for writing complex regular expressions with many capturing groups in a readable way.

At the moment, it contains just one very simple function (called `reagex`) and an utility function, but any function which could be useful for writing readable patterns is welcome.

Note: Publishing this ridiculously small project is an excuse to familiarize with python packaging, DevOps tools and the entire workflow behind the publication of an open-source project. The project template was generated using https://github.com/ionelmc/cookiecutter-pylibrary/ which is obviously an overkill for a "one-function-project".

- Free software: BSD 2-Clause License

1.1 Usage

The core function `reagex` is just a wrapper of `str.format` and it works in the same way. See the example

```
import re
from reagex import reagex
```

(continues on next page)

(continued from previous page)

```
# A sloppy pattern for an italian address (just to show how it works)
pattern = reagex(
    '{_address}, {postcode} {city} {province}',
    # groups starting with "_" are non-capturing
    _address = reagex(
        '{street} {number}',
        street = '(via|contrada|c/da|c[.]da|piazza|p[.]za|p[.]zza) [a-zA-Z]+',
        number = 'snc|[0-9]+'
    ),
    postcode = '[0-9]{5}',
    city = '[A-Za-z]+',
    province = '[A-Z]{2}'
)

matcher = re.compile(pattern)
match = matcher.fullmatch('via Roma 123, 12345 Napoli NA')
print(match.groupdict())

# prints:
#   {'city': 'Napoli',
#    'number': '123',
#    'postcode': '12345',
#    'province': 'NA',
#    'street': 'via Roma'}
```

Groups starting by `'_'` are non-capturing. The rest are all named capturing groups.

1.2 Why not...

1.2.1 Why not using just re.VERBOSE?

I think `reagex` is easier to write and to read:

- with reagex, you first describe the structure of the pattern in terms of groups, *then* you provide a pattern for each group; with re.VERBOSE you have to define the groups in the exact position they must be matched: to get the high-level structure of the pattern you may need to read multiple lines at the same indentation level
- with re.VERBOSE you just write a big string; with reagex you get syntax highlighting which helps readability
- white-spaces don't need any special treatment
- "{group_name}" is nicer than "(?P<group_name>)"

1.3 Installation

```
pip install reagex
```

1.4 Documentation

https://python-reagex.readthedocs.io/

1.5 Development

Possible improvements:

1. make some meaningful use of the `format_spec` in `{group_name:format_spec}`
2. add utility functions like `repeated` to help writing common patterns in a readable way

1.5.1 Testing

To run all the tests:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER 2

Installation

At the command line:

```
pip install reagex
```

CHAPTER 3

Reference

3.1 reagex

reagex.**reagex**(*pattern*, ***group_patterns*)

Utility function for writing regular expressions with many capturing groups in a readable, clean and hierarchical way. It is just a wrapper of `str.format` and it works in the same way. A minimal example:

```
pattern = reagex(
    '{name} "{nickname}" {surname}',
    name='[A-Z][a-z]+',
    nickname='[a-z]+',
    surname='[A-Z][a-z]+'
)
```

Parameters

- **pattern** (*str*) – a pattern where you can use `str.format` syntax for groups `{group_name}`. Groups are capturing unless they starts with `'_'`. For each group in this argument, this function expects a keyword argument with the same name containing the pattern for the group.
- ****group_patterns** (*str*) – patterns associated to groups; for each group in `pattern` of the kind `{group_name}` this function expects a keyword argument.

Returns a pattern you can pass to `re` functions

reagex.**repeated**(*pattern*, *sep*, *least=1*, *most=None*)

Returns a pattern that matches a sequence of strings that match `pattern` separated by strings that match `sep`.

For example, for matching a sequence of `'{key}={value}'` pairs separated by `'&'`, where key and value contains only lowercase letters:

```
repeated('[a-z]+=[a-z]+', '&') == '[a-z]+=[a-z]+(?:&[a-z]+=[a-z]+)*'
```

Parameters

- **pattern** (*str*) – a pattern
- **sep** (*str*) – a pattern for the separator (usually just a character/string)
- **least** (*int, positive*) – minimum number of strings matching `pattern`; must be positive
- **most** (*Optional[int]*) – maximum number of strings matching `pattern`; must be greater or equal to `least`

Returns a pattern

CHAPTER 4

Contributing

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

4.1 Bug reports

When 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.

4.2 Documentation improvements

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

4.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/janluke/python-reagex/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 code contributions are welcome :)

4.4 Development

To set up *python-reagex* for local development:

1. Fork python-reagex (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-reagex.git
```

3. Create a branch for local development:

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

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. 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
```

6. Submit a pull request through the GitHub website.

4.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

4.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 5

Authors

- Gianluca Gippetto

CHAPTER 6

Changelog

6.1 0.1.2 (2018-12-16)

- Fix little mistake in the example (which is showed in PyPI, so a release was necessary to update the PyPI page).

6.2 0.1.1 (2018-12-12)

- Minor fixes and modifications to documentation

6.3 0.1.0 (2018-12-08)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

R