repr

Release 0.3.0

June 20, 2016

Contents

1 Overview **1**
- 1.1 Installation . . . 1
- 1.2 Reasoning . . . 1
- 1.3 Usage . . . 2
- 1.4 Documentation . . . 2
- 1.5 Development . . . 2

2 Reference **5**
- 2.1 magic_repr . . . 5

3 Contributing **7**
- 3.1 Bug reports . . . 7
- 3.2 Documentation improvements . . . 7
- 3.3 Feature requests and feedback . . . 7
- 3.4 Development . . . 7

4 Authors **9**

5 Changelog **11**
- 5.1 0.3.0 (2016-06-20) . . . 11
- 5.2 0.2.1 (2016-06-19) . . . 11
- 5.3 0.2.0 (2016-06-19) . . . 11
- 5.4 0.1.0 (2016-06-09) . . . 11

6 Indices and tables **13**

Python Module Index **15**

CHAPTER 1

Overview

docs	
tests	
package	

A magic shortcut to generate __repr__ methods for your classes.

- Free software: BSD license

1.1 Installation

```
pip install repr
```

This package contains a single module `magic_repr` called so to not conflict with standart python's `repr`.

1.2 Reasoning

What do you think each time, writing such code?

```
def __repr__(self):
    return """
Issue(changelog={self.changelog},
      type={self.type},
      comment={self.comment},
      created_at={self.created_at},
      resolved_at={self.resolved_at})""".format(self=self).strip().encode('utf-8')
```

Isn't this much better and readable?

```
__repr__ = make_repr('changelog', 'type', 'comment', 'created_at', 'resolved_at')
```

And this produces much nicer output:

```
<Issue changelog=<Changelog namespace=u'python'
                            name=u'geocoder'
                            source=u'https://github.com/DenisCarriere/geocoder'>
       type=u'wrong-version-content'
```

```
       comment=u'AllMyChanges should take release notes from the web site.'
       created_at=datetime.datetime(2016, 6, 17, 6, 44, 52, 16760, tzinfo=<UTC>)
       resolved_at=None>
```

1.2.1 Another advantage of the magic_repr

Is it's recursiveness. If you use `magic_repr` for your objects and they include each other, then representation of the parent object will include child objects properly nested:

```
<Foo bars={1: <Bar first=1
                  second=2
                  third=3>,
           2: <Bar first=1
                  second=2
                  third=3>,
           u'': <Bar first=1
                     second=2
                     third=3>}>
```

And all this for free! Just do `__repr__ = make_repr()`.

1.3 Usage

For simple cases it is enough to call `make_repr` without arguments. It will figure out which attributes object has and will output them sorted alphabetically.

You can also specify which attributes you want to include in "representaion":

```
__repr__ = make_repr('foo', 'bar')
```

And to specify a function to create a value for an attribute, using keywords:

```
class Some(object):
    def is_active(self):
        return True

Some.__repr__ = make_repr(active=Some.is_active)
```

Pay attention, that in this case `__repr__` was created after the class definition. This is because inside of the class it can't reference itself.

1.4 Documentation

https://python-repr.readthedocs.org/

1.5 Development

To run the all tests run:

```
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

Reference

2.1 magic_repr

magic_repr.**make_repr**(**args*, ***kwargs*)

Returns __repr__ method which returns ASCII representaion of the object with given fields.

Without arguments, `make_repr` generates a method which outputs all object's non-protected (non-undercored) arguments which are not callables.

Accepts `*args`, which should be a names of object's attributes to be included in the output:

```
    __repr__ = make_repr('foo', 'bar')
```

If you want to generate attribute's content on the fly, then you should use keyword arguments and pass a callable of one argument:

```
    __repr__ = make_repr(foo=lambda obj: obj.blah + 100500)
```

CHAPTER 3

Contributing

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

3.1 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.

3.2 Documentation improvements

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

3.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/svetlyak40wt/python-repr/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 :)

3.4 Development

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

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

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

3.4.1 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`.

3.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -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 4

Authors

- Alexander Artemenko - http://dev.svetlyak.ru

CHAPTER 5

Changelog

5.1 0.3.0 (2016-06-20)

- Now `make_repr` can be used for recursive datastructures.

5.2 0.2.1 (2016-06-19)

- Documentation improved.

5.3 0.2.0 (2016-06-19)

- Better handling of nested datastructure.
- Callables as source of the attribute's value.
- Some documentation.

5.4 0.1.0 (2016-06-09)

- First release on PyPI.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

m

magic_repr, 5

M

magic_repr (module), 5
make_repr() (in module magic_repr), 5