@environment Documentation

Release 0.2.1

Alexandr Mansurov

Apr 01, 2019

Contents:

CHAPTER 1

@environment

Decorator for convenient loading of environment variables. @environment allows you to declare dependencies on environment variables so that it's clear what needs to be set. Also, any error handling is moved away making the code clearer.

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

1.1 Getting started

Install @environment from pip:

```
pip install atenvironment
```

Using @environment is as simple as:

```
from atenvironment import environment

@environment('API_KEY', 'TOKEN')
def check(a, b, c, key, token):
    # API_KEY is in key
    # TOKEN is in token
```

Then call the function as:

```
check(a, b, c)
```

Environment variables are checked and provided to the function as trailing parameters in order of declaration. In case the token is not in environment an `atenvironment.EnvironMiss` exception is raised. You can also provide your own error handling function. In addition, some environment variables can be loaded directly into object variable in case instance property is to be initialized.

See the documentation for more details.

CHAPTER 2

Installation

2.1 Stable release

To install @environment, run this command in your terminal:

```
$ pip install atenvironment
```

This is the preferred method to install @environment, 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 @environment can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/eghuro/atenvironment
```

Or download the tarball:

```
$ curl  -OL https://github.com/eghuro/atenvironment/tarball/master
```

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

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

CHAPTER 3

Usage

To use @environment in a project:

```
from atenvironment import environment
```

CHAPTER 4

atenvironment

4.1 atenvironment package

4.1.1 Module contents

Top-level package for @environment.

exception atenvironment.**DecoratorSyntaxError**
Bases: BaseException

exception atenvironment.**EnvironMiss**
Bases: KeyError

Error indicating key is not present in environment.

exception atenvironment.**UnknownKeyword**
Bases: BaseException

Exception indicating unknown keyword was provided to @environment decorator in **kwargs.

atenvironment.**environment**(*value*, ***kwargs*)
@environment decorator.

Arguments: value – one or more environment tokens requested

onerror – optional function to be called if any of the environment tokens in value is not present in environment. Such function must take one parameter what is a string value of a missing environment token. If onerror is not set, error is logged and EnvironMiss exception is raised

in_self – optional list of variable names in case instance property is to be initialized. If present, must be of same length as value. None in this list means particular element will be passed into the function and not into any instance property.

default – optional list of default values in case environment token in value is not present in environment. If present, must be of same length as value. onerror will not be called in such case.

The decorator checks for presence of environment tokens and if successful reads their values to the function parameters of the decorated function after any called parameters provided.

Eg. if calling function(a, b, c) that is decorated with @environment('X') the function must be defined as def function(a, b, c, x) and X from the environment is read as last parameter.

When combining decorators or using multiple environment tokens in one @environment('X', 'Y', 'Z') the values are loaded from the left to the right, from the top to the bottom.

If a function parameter for @environment is missing, when such function is called a TypeError is raised by the interpreter.

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/eghuro/atenvironment/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

@environment could always use more documentation, whether as part of the official @environment 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/eghuro/atenvironment/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 *atenvironment* for local development.

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

```
$ git clone git@github.com:your_name_here/atenvironment.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 atenvironment
$ cd atenvironment/
$ 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 atenvironment 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 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/eghuro/atenvironment/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_atenvironment
```

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

- Alexandr Mansurov <alex@eghuro.cz>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.0 (2018-06-24)

- on error: optional function to be called if any of the environment tokens in value is not present in environment
- in self: optional variable name in case instance property is to be initialized
- package slightly restructured

7.2 0.1.0 (2018-06-12)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index