Deprecated Documentation

Release 1.1.0

Marcos CARDOSO and Laurent LAPORTE

Nov 20, 2017

Contents

Welcome to Deprecated's Documentation. This documentation is divided into different parts. I recommend that you get started with *Installation* and then head over to the *Tutorial*. If you'd rather dive into the internals of the Deprecated Library, check out the *API* documentation.

CHAPTER 1

User's Guide

This part of the documentation, which is mostly prose, begins with some background information about Deprecated, then focuses on step-by-step instructions for using Deprecated.

1.1 Installation

1.1.1 Python Version

We recommend using the latest version of Python 3. Deprecated Library supports Python 3.3 and newer, Python 2.6 and newer, and PyPy.

1.1.2 Dependencies

This library has no dependency (except development dependencies).

Development dependencies

These distributions will not be installed automatically. You need to install them explicitly with *pip install -e .[dev]*.

- pytest is a framework which makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries...
- pytest-cov is a pytest plugin used to produce coverage reports.
- tox aims to automate and standardize testing in Python. It is part of a larger vision of easing the packaging, testing and release process of Python software...
- bumpversion is a small command line tool to simplify releasing software by updating all version strings in your source code by the correct increment. Also creates commits and tags...
- sphinx is a tool that makes it easy to create intelligent and beautiful documentation.

1.1.3 Virtual environments

Use a virtual environment to manage the dependencies for your project, both in development and in production.

What problem does a virtual environment solve? The more Python projects you have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project.

Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages.

Python 3 comes bundled with the `venv` module to create virtual environments. If you're using a modern version of Python, you can continue on to the next section.

If you're using Python 2, see *Install virtualenv* first.

Create an environment

Create a project folder and a `venv` folder within:

```
mkdir myproject
cd myproject
python3 -m venv venv
```

On Windows:

```
py -3 -m venv venv
```

If you needed to install virtualenv because you are on an older version of Python, use the following command instead:

```
virtualenv venv
```

On Windows:

```
\Python27\Scripts\virtualenv.exe venv
```

Activate the environment

Before you work on your project, activate the corresponding environment:

```
. venv/bin/activate
```

On Windows:

```
venv\Scripts\activate
```

Your shell prompt will change to show the name of the activated environment.

1.1.4 Install Deprecated

Within the activated environment, use the following command to install Deprecated:

```
pip install Deprecated
```

Living on the edge

If you want to work with the latest Deprecated code before it's released, install or update the code from the master branch:

```
pip install -U https://github.com/tantale/deprecated/archive/master.tar.gz
```

1.1.5 Install virtualenv

If you are using Python 2, the venv module is not available. Instead, install virtualenv.

On Linux, virtualenv is provided by your package manager:

```
# Debian, Ubuntu
sudo apt-get install python-virtualenv

# CentOS, Fedora
sudo yum install python-virtualenv

# Arch
sudo pacman -S python-virtualenv
```

If you are on Mac OS X or Windows, download get-pip.py, then:

```
sudo python2 Downloads/get-pip.py
sudo python2 -m pip install virtualenv
```

On Windows, as an administrator:

```
\Python27\python.exe Downloads\get-pip.py
\Python27\python.exe -m pip install virtualenv
```

Now you can continue to *Create an environment*.

1.2 Tutorial

In this tutorial, we will use the Deprecated Library to mark pieces of codes as deprecated. We will also see what's happened when a user tries to call deprecated codes.

1.2.1 Deprecated function

First, we have this little library composed of a single module: `liberty.py`:

```
# coding: utf-8
""" Liberty library is free """

import pprint


def print_value(value):
    """
    Print the value
```

```
    :param value: The value to print
    """
    pprint.pprint(value)
```

You decided to write a more powerful function called `better_print()` which will become a replacement of `print_value()`. And you decided that the later function is deprecated.

To mark the `print_value()` as deprecated, you can use the *deprecated()* decorator:

```
# coding: utf-8
""" Liberty library is free """

import pprint

from deprecated import deprecated


@deprecated
def print_value(value):
    """
    Print the value

    :param value: The value to print
    """
    pprint.pprint(value)


def better_print(value, printer=None):
    """
    Print the value using a *printer*.

    :param value: The value to print
    :param printer: Callable used to print the value, by default: :func:`pprint.
↪pprint`
    """
    printer = printer or pprint.pprint
    printer(value)
```

If the user tries to use the deprecated functions, he will have a warning for each call:

```
# coding: utf-8
import liberty

liberty.print_value("hello")
liberty.print_value("hello again")
liberty.better_print("Hi Tom!")
```

```
$ python use_liberty.py

using_liberty.py:4: DeprecationWarning: Call to deprecated function print_value.
  liberty.print_value("hello")
'hello'
using_liberty.py:5: DeprecationWarning: Call to deprecated function print_value.
  liberty.print_value("hello again")
'hello again'
'Hi Tom!'
```

As you can see, the deprecation warning is displayed like a stack trace. You have the source code path and line number

and the called function. This is very useful for debugging. But, this doesn't help the developer to choose a alternative: which function could he use instead?

To help the developer, you can add a "reason" message. For instance:

```
# coding: utf-8
""" Liberty library is free """

import pprint

from deprecated import deprecated


@deprecated("This function is rotten, use 'better_print' instead")
def print_value(value):
    """
    Print the value

    :param value: The value to print
    """
    pprint.pprint(value)


def better_print(value, printer=None):
    """
    Print the value using a *printer*.

    :param value: The value to print
    :param printer: Callable used to print the value, by default: :func:`pprint.
↪pprint`
    """
    printer = printer or pprint.pprint
    printer(value)
```

When the user calls the deprecated functions, he will have a more useful message:

```
$ python use_liberty.py

using_liberty.py:4: DeprecationWarning: Call to deprecated function print_value (This␣
↪function is rotten, use 'better_print' instead).
  liberty.print_value("hello")
'hello'
using_liberty.py:5: DeprecationWarning: Call to deprecated function print_value (This␣
↪function is rotten, use 'better_print' instead).
  liberty.print_value("hello again")
'hello again'
'Hi Tom!'
```

1.2.2 Deprecated method

Decorating a deprecated method works like decorating a function.

```
# coding: utf-8
""" Liberty library is free """

import pprint
```

```
from deprecated import deprecated


class Liberty(object):
    def __init__(self, value):
        self.value = value

    @deprecated("This method is rotten, use 'better_print' instead")
    def print_value(self):
        """ Print the value """
        pprint.pprint(self.value)

    def better_print(self, printer=None):
        """
        Print the value using a *printer*.

        :param printer: Callable used to print the value, by default: :func:`pprint.
↪pprint`
        """
        printer = printer or pprint.pprint
        printer(self.value)
```

When the user calls the deprecated methods, like this:

```
# coding: utf-8
import liberty

obj = liberty.Liberty("Greeting")
obj.print_value()
obj.print_value()
obj.better_print()
```

He will have:

```
$ python use_liberty.py

using_liberty.py:5: DeprecationWarning: Call to deprecated function print_value (This␣
↪method is rotten, use 'better_print' instead).
  obj.print_value()
'Greeting'
using_liberty.py:6: DeprecationWarning: Call to deprecated function print_value (This␣
↪method is rotten, use 'better_print' instead).
  obj.print_value()
'Greeting'
'Greeting'
```

Note: The call is done to the same object, so we have 3 times 'Greeting'.

1.2.3 Deprecated class

You can also decide that a class is deprecated.

For instance:

```
# coding: utf-8
""" Liberty library is free """

import pprint

from deprecated import deprecated


@deprecated("This class is not perfect")
class Liberty(object):
    def __init__(self, value):
        self.value = value

    def print_value(self):
        """ Print the value """
        pprint.pprint(self.value)
```

When the user use the deprecated class like this:

```
# coding: utf-8
import liberty

obj = liberty.Liberty("Salutation")
obj.print_value()
obj.print_value()
```

He will have a warning at object instantiation. Once the object is initialised, no more warning are emitted.

```
$ python use_liberty.py

using_liberty.py:4: DeprecationWarning: Call to deprecated class Liberty (This class␣
↪is not perfect).
  obj = liberty.Liberty("Salutation")
'Salutation'
'Salutation'
```

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API

This part of the documentation covers all the interfaces of the Deprecated Library.

2.1.1 Deprecated decorator

Deprecated Library

Python `@deprecated` decorator to deprecate old python classes, functions or methods.

`deprecated.`**`deprecated`**(*reason*)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

Classic usage:

To use this, decorate your deprecated function with **@deprecated** decorator:

```
from deprecated import deprecated


@deprecated
def some_old_function(x, y):
    return x + y
```

You can also decorate a class or a method:

```
from deprecated import deprecated
```

```
class SomeClass(object):
    @deprecated
    def some_old_method(self, x, y):
        return x + y


@deprecated
class SomeOldClass(object):
    pass
```

You can give a "reason" message to help the developer to choose another function/class:

```
from deprecated import deprecated


@deprecated(reason="use another function")
def some_old_function(x, y):
    return x + y
```

Parameters **reason** (*str or callable or type*) – Reason message (or function/class/method to decorate).

CHAPTER 3

Additional Notes

Legal information and changelog are here for the interested.

3.1 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

Note: The library **"Python-Deprecated"** was renamed **"Deprecated"**, simply! This project is more consistent because now, the name of the library is the same as the name of the Python package.

- In your `setup.py`, you can replace the "Python-Deprecated" dependency with "Deprecated".
- In your source code, nothing has changed, you will always use `import deprecated`, as before.
- I decided to keep the same version number because there is really no change in the source code (only in comment or documentation).

3.1.1 v1.1.0 (2017-11-06)

Minor release

Added

- Change in *`deprecated.deprecated()`* decorator: you can give a "reason" message to help the developer choose another class, function or method.
- Add support for Universal Wheel (Python versions 2.6, 2.7, 3.3, 3.4, 3.5, 3.6 and PyPy).
- Add missing `__doc__` and `__version__` attributes to *`deprecated`* module.

- Add an extensive documentation of Deprecated Library.

Other

- Improve Travis configuration file (compatibility from Python 2.6 to 3.7-dev, and PyPy).
- Add AppVeyor configuration file.
- Add Tox configuration file.
- Add BumpVersion configuration file.
- Improve project settings: add a long description for the project. Set the **license** and the **development status** in the classifiers property.
- Add the `CONTRIBUTING.rst` file: "How to contribute to Deprecated Library".

3.1.2 v1.0.0 (2016-08-30)

Major release

Added

- **deprecated**: Created **@deprecated** decorator

3.2 License

3.3 How to contribute to Deprecated Library

Thank you for considering contributing to Deprecated!

3.3.1 Support questions

Please, don't use the issue tracker for this. Use one of the following resources for questions about your own code:

- Ask on Stack Overflow. Search with Google first using: `site:stackoverflow.com deprecated decorator {search term, exception message, etc.}`

3.3.2 Reporting issues

- Describe what you expected to happen.
- If possible, include a minimal, complete, and verifiable example to help us identify the issue. This also helps check that the issue is not with your own code.
- Describe what actually happened. Include the full traceback if there was an exception.
- List your Python, Deprecated versions. If possible, check if this issue is already fixed in the repository.

3.3.3 Submitting patches

- Include tests if your patch is supposed to solve a bug, and explain clearly under which circumstances the bug happens. Make sure the test fails without your patch.
- Try to follow PEP8, but you may ignore the line length limit if following it would make the code uglier.

First time setup

- Download and install the latest version of git.
- Configure git with your username and email:

```
git config --global user.name 'your name'
git config --global user.email 'your email'
```

- Make sure you have a GitHub account.
- Fork Deprecated to your GitHub account by clicking the Fork button.
- Clone your GitHub fork locally:

```
git clone https://github.com/{username}/deprecated.git
cd deprecated
```

- Add the main repository as a remote to update later:

```
git remote add tantale https://github.com/tantale/deprecated.git
git fetch tantale
```

- Create a virtualenv:

```
python3 -m venv env
. env/bin/activate
# or "env\Scripts\activate" on Windows
```

- Install Deprecated in editable mode with development dependencies:

```
pip install -e ".[dev]"
```

Start coding

- Create a branch to identify the issue you would like to work on (e.g. `2287-dry-test-suite`)
- Using your favorite editor, make your changes, committing as you go.
- Try to follow PEP8, but you may ignore the line length limit if following it would make the code uglier.
- Include tests that cover any code changes you make. Make sure the test fails without your patch. *Running the tests*.
- Push your commits to GitHub and create a pull request.
- Celebrate

Running the tests

Run the basic test suite with:

```
pytest tests/
```

This only runs the tests for the current environment. Whether this is relevant depends on which part of Deprecated you're working on. Travis-CI will run the full suite when you submit your pull request.

The full test suite takes a long time to run because it tests multiple combinations of Python and dependencies. You need to have Python 2.6, 2.7, 3.3, 3.4, 3.5 3.6, and PyPy 2.7 installed to run all of the environments. Then run:

```
tox
```

Running test coverage

Generating a report of lines that do not have test coverage can indicate where to start contributing. Run `pytest` using `coverage` and generate a report on the terminal and as an interactive HTML document:

```
pytest --cov-report term-missing --cov-report html --cov=deprecated tests/
# then open htmlcov/index.html
```

Read more about coverage.

Running the full test suite with `tox` will combine the coverage reports from all runs.

`make` targets

Deprecated provides a `Makefile` with various shortcuts. They will ensure that all dependencies are installed.

- `make test` runs the basic test suite with `pytest`
- `make cov` runs the basic test suite with `coverage`
- `make test-all` runs the full test suite with `tox`

Generating the documentation

The documentation is automatically generated with ReadTheDocs for each git push on master. You can also generate it manually using Sphinx.

To generate the HTML documentation, run:

```
sphinx-build -b html docs/source/ dist/docs/html/
```

To generate the epub v2 documentation, run:

```
sphinx-build -b epub2 docs/source/ dist/docs/epub/
```

Python Module Index

d

Index

D