wabisabi Documentation

Release 0.2.8+5.g0116dc4.dirty

Mark Harfouche

Oct 30, 2022

CONTENTS:

wabisabi

[![pypi](https://img.shields.io/pypi/v/wabisabi.svg){}](https://pypi.python.org/pypi/wabisabi) [![Travis](https://img.shields.io/travis/hmaarrfk/wabisabi.svg){}](https://travis-ci.org/hmaarrfk/wabisabi) [![Docs](https://readthedocs.org/projects/wabisabi/badge/?version=latest){}](https://wabisabi.readthedocs.io/en/latest/?badge=latest)

Python3 wabisabi. Automatically write boilerplate code for many kinds of deperecations through python decorators.

- Free software: [BSD license](LICENSE)
- [Documentation](https://wabisabi.readthedocs.io)

Link above not working for now: https://deprecation-factory.readthedocs.io/en/latest/?badge=latest

Motivations Breaking things is important! Breaking other's things is just mean!

The goal of deprecations is to warn other library writers that their code is about to break so you can keep making agressive changes to your own.

Often when you want to deprecate a feature, you end up following a procedure similar to

1. Make the useful modification to your code.
2. Decide on when the old behaviour should be switched over.
3. Add warnings INSIDE your function to warn users.
4. Change the function signature to something non-sensical to detect the default behaviour.
5. Add messages in the documentation (numpydoc compatible)

Finally, when the behaviour is official deprepreprecated, you need to do all these changes again.

6. Remove the warnings.
7. Remove the documentation messages.
8. Remove the old behaviour.
9. Change the function signature back to something useful.

The goal of this library is to allow you to shortcut steps 3-9. You shouldn't have to revisit the deprecation long after you completed implementing your new features. You write your code how it is **supposed to look**, this library, makes ensures your users have enough time to update their code.

This library

- Modifies function signatures so to ensure correctness for the current version. This should help with autocompletions.
- Adds a warning section to the docstrings. An attempt is made to properly indent the docstring.
- Point the user to **their** line of code, so that they know where to make make the appropriate modification.
- Leaving deprecators in place after the desired threshold results in a noop. This means that you can be lazy about ripping them out of code. Deprecations should not have to be blockers for your development.
- If numpydoc > 0.7 is installed, the "Warns" sections are combined into a single section allowing you to chain deprecators.

Installation

While you can depend on this, I strongly recommend you version the files you need in your project as the API is highly likely to change and break your code.

Current deprecators

- Deprecator for change of default values in *kwargs*. Handles *kwargs* passed as positional arguments too!

- Transitionning to keyword only arguments.
- *kwarg* renaming

Future deprecators

- Swapping the order of positional arguments
- Making an old *kwarg* a manditory positional *arg*
- Feature requests are welcome!

Development Lead

- Mark Harfouche

Contributors

Could be you!

How to contribute Ready to contribute? We use the standard github contribution model. scikit-image has a great [writeup](http://scikit-image.org/docs/dev/contribute.html) on how to setup your environment. Adapt it for our environment.

Cookiecutter

This package was created with [Cookiecutter](https://github.com/audreyr/cookiecutter) and the [audreyr/cookiecutter-pypackage](https://github.com/audreyr/cookiecutter-pypackage) project template.

CHAPTER

ONE

INSTALLATION

Deprecation Factory is meant to be a project dependency and not installed directly by the end user. As such, we assume that you know how to manage depdencies for your project.

Deprecation Factory only depends on the Python standard library keeping it lightweight. This package is available both on PyPi and conda-forge.

CHAPTER

TWO

USAGE

This project is meant to be used by library developers. You are meant to use `functools.partial` as a shortecuct to set things like your friendly library name, and the current library version.

In something like your project's `utils` file, you should include adapted versions of this code:

```
from wabisabi import kwonly_change
from wabisabi import default_parameter_change

from functools import partial
from ..__init__ import __version__  # or the applicable line

default_parameter_change = partial(default_parameter_change,
                                   current_library_version=__version__,
                                   library_name='mylib')
kwonly_change = partial(kwonly_change,
                        library_name='my super lib',
                        current_library_version=__version__)
```

After this, you can use the decorators to deprecated functions how you please

CHAPTER

THREE

WABISABI

wabisabi.**default_parameter_change**(*version*, *old_kwargs*, *library_name*, *current_library_version*)

Deprecates a default value for a kwarg.

If the software version is greater or equal to that of `version`, this decorator returns the original function without any modifications.

If the user doesn't specify a parameter that will change default values, it will issue a `FutureWarning` pointing to the line of his code, in a single warning for all changes in the default parameters.

It also modifies the `__signature__` of the function so that the function appears to have the default parameters to whatever the default values are in the current version of the function.

Finally, if a docstring is provided, it appends a warning message compatible numpydoc.

Parameters

version: version-like
The version in which the parameter will take the new default value. If the software version reaches this value, the new default will be taken on without warning.

old_kwargs:
The list of keyword arguments, with their old default values specified, as a dictionary. You should be able to copy paste your old keyword arguments and put then in a *dict*.

library_name
[str] The human readable name for your library.

current_library_version
[version-like] The current version of your library.

Examples

```
>>> @default_parameter_change('0.16', doc(this='tim'))
... def foo(this='foo', bar='zip'):
...     '''Prints your parameter.
...
...     **This function has a depeprecation decorator set to '0.16'**
...
...     Parameters
...     ----------
...     this : str
...         This is the string parameter that should be printed
...
```

(continues on next page)

(continued from previous page)

```
...     '''
...     print(bar)
...     return this
```

wabisabi.**kwonly_change**(*version*, *previous_arg_order=None*, *keep_old_signature=False*, *library_name=None*, *current_library_version=None*)

Returns a decorator that enforces a smaller number of positional arguments.

If the current library version is smaller than version in which the new signature takes effect, this deprecator will:

Issue a warnign pointing to the user's code if they call a function with too many positional arguments.

If you want to move the new keyword only arguments after the keyword only seperator, you should specify a list with the previous order of arguments as the parameter `previous_arg_order`. If this parameter is not specified, it is assumed that the previous function signature allowed all parameters to be specified as either positional or keyword arguments.

Parameters

version: version-like
Version in which the new function signature takes full effect.

previous_arg_order: list of strings or ``None``
If the function previously had keyword only arguments, you should use this to specify the names of previous positional arguments. If set to none, it is assumed that the previous version of the function would accept all parameters as positional or keyword arguments. Specifying this also allows you to re-order the new keyword only. To re-iterate, you cannot swap the order of positional arguments with this deprecator.

keep_old_signature: bool
If set to true, the signature will reflect the previous signature of the function. This is not recommended since showing new users the keyword-only version of the signature will not cause them to write wrong code. If they follow the new signature, their code will be correct and they will avoid the `FutureWarning`.

library_name: str
Friendly library name to include in warning messages

current_library_version: version-like
The current version of the shipped library (typically your module's `__version__`).

Examples

```
>>> from wabisabi import kwonly_change
>>> import functools
>>>
>>> kwonly_change = functools.partial(kwonly_change,
...                                   library_name='my super lib',
...                                   current_library_version='0.14')
>>> @kwonly_change('0.15')
... def foo_new(zap='zip', *, bar='hello', baz='world'):
...     return zap + bar + baz
```

CHAPTER

FOUR

USAGE EXAMPLES

4.1 Argument default value

One important advantage of using these decorators is that they ensure that the docstrings are correct. This module has as hypothetical version number of `0.14`. Click on the source links to see what the source looks like for each of these functions.

Look at the doc page to see how your users would see your documetnation.

`default_value.foo`(*this='that'*)

Prints your parameter.

This function has no deprecation decorator.

Parameters

this
[str] This is the string parameter that should be printed

`default_value.foo_deprecated`(*two=2*, *this='tim'*, *one=1*)

Prints your parameter.

This function has a depeprecation decorator set to '0.16'

Parameters

this
[str] This is the string parameter that should be printed

Warns

FutureWarning
In release 0.16 of my super lib, this function will take on new default value(s) for the following keyword argument(s):

this : *'tim'* -> *'that'*

To avoid this warning in your code, specify the value of all listed keyword arguments.

`default_value.foo_deprecated_13`(*this='that'*)

Prints your parameter

This function has a deprecation decorator set to '0.13'.

Parameters

this
[str] This is the string parameter that should be printed.

default_value.**foo_two_params**(*bar='hello'*, *baz='world'*)

Joins two strings with a space between them.

This function has a no deprecation decorator.

Parameters

bar
[str] The first word to join.

baz
[str] The second word to join.

default_value.**foo_two_params_deprecating**(*bar='bonjour'*, *baz='monde'*)

Joins two strings with a space between them.

This function has a deprecation decorator set to '0.16'.

Parameters

bar
[str] The first word to join.

baz
[str] The second word to join.

Warns

FutureWarning
In release 0.16 of my super lib, this function will take on new default value(s) for the following keyword argument(s):

bar : *'bonjour'* -> *'hello'*

baz : *'monde'* -> *'world'*

To avoid this warning in your code, specify the value of all listed keyword arguments.

default_value.**foo_two_params_redux**(*bar='bonjour'*, *baz='monde'*)

Joins two strings with a space between them.

This function has a no deprecation decorator.

Parameters

bar
[str] The first word to join.

baz
[str] The second word to join.

Warns

FutureWarning
In release 0.16 of my super lib, this function will take on new default value(s) for the following keyword argument(s):

bar : *'bonjour'* -> *'hello'*

To avoid this warning in your code, specify the value of all listed keyword arguments.

FutureWarning
In release 0.17 of my super lib, this function will take on new default value(s) for the following keyword argument(s):

baz : *'monde'* -> *'world'*

To avoid this warning in your code, specify the value of all listed keyword arguments.

4.2 Keyword only deprecation

This is how the docstrings will appear if you decide to deprecate positional arguments to keyword.

`kwonly.foo_new`(*zap='zip'*, ***, *bar='hello'*, *baz='world'*)

Adds bar and baz

The decorator will infer the old order of the paramters.

Parameters

bar: str
a string

baz: str
an other python object.

zap: str
zaps everything to dust

Returns

result: str
the sum of the zapping.

Warns

FutureWarning
In release 0.15 of my super lib, the argument(s):

bar, baz

will become keyword-only arguments. To avoid this warning, provide all the above arguments as keyword arguments.

`kwonly.foo_new_keep_signature`(*zap='zip'*, *bar='hello'*, *baz='world'*)

Adds bar and baz

You can choose to keep the old signature.

Parameters

bar: str
a string

baz: str
an other python object.

zap: str
zaps everything to dust

Returns

result: str
the sum of the zapping.

Warns

FutureWarning
In release 0.15 of my super lib, the argument(s):

bar, baz

will become keyword-only arguments. To avoid this warning, provide all the above arguments as keyword arguments.

kwonly.**foo_new_swap_arguments**(*zap='zip'*, ***, *baz='world'*, *bar='hello'*)

Adds bar and baz

You can even swap the order of the arguments, though you should specify how they were ordered.

Parameters

bar: str
a string

baz: str
an other python object.

zap: str
zaps everything to dust

Returns

result: str
the sum of the zapping.

Warns

FutureWarning
In release 0.15 of my super lib, the argument(s):

bar, baz

will become keyword-only arguments. To avoid this warning, provide all the above arguments as keyword arguments.

kwonly.**foo_old**(*zap='zip'*, *bar='hello'*, *baz='world'*)

Adds bar and baz

Parameters

bar: str
a string

baz: str
an other python object.

zap: str
zaps everything to dust

Returns

result: str
the sum of the zapping.

History

0.2.4 (2018.08.13) * Apparently numpydoc 0.6 didn't have a __*version*__ attribute....

0.2.3 (2018.08.13)

- Check for numpydoc 0.7
- Fix a typo in the docstring message

0.2.2 (2018.08.12)

- API change. change_default_parameter now takes a dictionary for the old_kwargs so that parameter names don't conflict

0.2.1 (2018.08.12)

- Merge with other numpydocs so that documentation in Sphinx doesn't crash

0.2.0 (2018.08.11)

- Provide a deprecator for changing the number of keyword only arguments.

0.1.1 (2018.08.09)

- Deprecated arguments appear in order for Python 3.5 as well.

0.1.0 (2018.08.09)

- New deprecator for changing the default value of *kwards*. Handles arguments passed as positional argumnets too.

0.0.1 (2018-07-29)

- First release on PyPi

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX