positional Documentation

Release 1.1.3.dev1

Morgan Fainberg <morgan.fainberg@gmail.com>, Jamie Lennox <

Jul 27, 2017

Contents

Contents:

CHAPTER 1

Modules

positional package

Module contents

class positional.**positional**(*max_positional_args=None*, *enforcement='except'*)

Bases: object

A decorator to enforce passing arguments as keywords.

When you have a function that takes a lot of arguments you expect people to pass those arguments as keyword arguments. Python however does not enforce this. In future then if you decide that you want to insert a new argument or rearrange the arguments or transition to using **kwargs you break compatibility with users of your code who (wrongly) gave you 20 positional arguments.

In python 3 there is syntax to prevent this however we are not all in the position where we can write python 3 exclusive code. Positional solves the problem in the mean time across both pythons by enforcing that certain arguments must be past as keyword arguments.

Parameters **max_positional_args** – the maixmum number of arguments that can be passed to this function without keyword parameters. Defaults to enforcing that every parameter with a default value must be passed as a keyword argument.

:type max_positional_args int

Parameters **enforcement** – defines the way incorrect usage is reported. Currenlty accepts *positional.EXCEPT* to raise a TypeError or *positional.WARN* to show a warning. A warning can be useful for applying to functions that are already public as a deprecation notice. Defaults to *positional.EXCEPT*.

EXCEPT = 'except'

WARN = 'warn'

classmethod **classmethod**(**args*, ***kwargs*)

classmethod **method**(*max_positional_args=None*, *enforcement='except'*)

CHAPTER 2

Usage

positional

A decorator which enforces only some args may be passed positionally. This library is minimally maintained and should only be used in cases of Python 2 to Python 3 conversions. Please write only Python 3 code going forward.

CHAPTER 3

The Basics

positional provides a decorator which enforces only some args may be passed positionally. The idea and some of the code was taken from the oauth2 client of the google-api client.

The decorator makes it easy to support Python 3 style key-word only parameters. For example, in Python 3 it is possible to write:

```
>>> def fn(pos1, *, kwonly1, kwonly2=None):
...     ...
```

All named parameters after * must be a keyword:

```
>>> fn(10, 'kw1', 'kw2')  # Raises exception.
>>> fn(10, kwonly1='kw1', kwonly2='kw2')  # Ok.
```

To replicate this behaviour with the positional decorator you simply specify how many arguments may be passed positionally.

First to import the decorator we typically use:

```
>> from positional import positional
```

Replicating the Example above:

```
>>> @positional(1)
... fn(pos1, kwonly1=None, kwonly2=None):
...     ...
```

If no default value is provided to a keyword argument, it becomes a required keyword argument:

```
>>> @positional(0)
... def fn(required_kw):
...     ...
```

This must be called with the keyword parameter:

```
>>> fn() # Raises exception
>>> fn(10) # Raises Exception
>>> fn(required_kw=10) # OK
```

When defining instance or class methods always remember that in python the first positional argument passed is the instance; you will need to account for *self* and *cls*:

```
>>> class MyClass(object):
...
...     @positional(2)
...     def my_method(self, pos1, kwonly1=None):
...         ...
...
...     @classmethod
...     @positional(2)
...     def my_method(cls, pos1, kwonly1=None):
...         ...
```

If you would prefer not to account for *self* and *cls* you can use the *method* and *classmethod* helpers which do not consider the initial positional argument. So the following class is exactly the same as the one above:

```
>>> class MyClass(object):
...
...     @positional.method(1)
...     def my_method(self, pos1, kwonly1=None):
...         ...
...
...     @positional.classmethod(1)
...     def my_method(cls, pos1, kwonly1=None):
...         ...
```

If a value isn't provided to the decorator then it will enforce that every variable without a default value will be required to be a kwarg:

```
>>> @positional()
... def fn(pos1, kwonly1=None):
...     ...
...
>>> fn(10)  # Ok.
>>> fn(10, 20)  # Raises exception.
>>> fn(10, kwonly1=20)  # Ok.
```

This behaviour will work with the *positional.method* and *positional.classmethod* helper functions as well:

```
>>> class MyClass(object):
...
...    @positional.classmethod()
...    def my_method(cls, pos1, kwonly1=None):
...        ...
...
>>> MyClass.my_method(10)  # Ok.
>>> MyClass.my_method(10, 20)  # Raises exception.
>>> MyClass.my_method(10, kwonly1=20)  # Ok.
```

For compatibility reasons you may wish to not always raise an exception so a WARN mode is available. Rather than raise an exception a warning will be emitted.

```
>>> @positional(1, enforcement=positional.WARN):
... def fn(pos1, kwonly=1):
...     ...
```

Available modes are:

- positional.EXCEPT - the default, raise an exception.
- positional.WARN - emit a warning.

CHAPTER 4

Release Notes

CHANGES

- Update README.rst

1.1.2

- Update setup.cfg
- Update .travis.yml
- Fix deprecation warning in Python 3 for inspect.getargspec()
- Update .travis.yml
- Update .travis.yml
- Update .travis.yml
- Update setup.cfg
- Update tox.ini
- Update .travis.yml
- update classifiers

1.1.1

- Update requirements.txt for pbr versions to match
- Fix the python3 demo in README

1.1.0

- Fix to preserve argspec

1.0.1

- Fix version string

1.0.0

- Add preferred way to import to README
- Emit a DeprecationWarning rather than log a message
- Stop ignoring doc/source/api
- Add Sphinx Documentation
- Move doc into readme and write shorter class doc
- Add proper info to README.rst
- Use from positional import positional syntax
- Remove references to gate
- Remove copied copyright
- Move back to .RST for README
- Update ignored flake8 error docs
- Add documentation badge for RTD
- Remove some openstack-specific cruft from .gitignore
- Add Travis-CI status to readme
- Add tests and support files
- Initial Commit

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index