tolerance Documentation

Release

Alisue

January 21, 2014

CONTENTS

CHAPTER

ONE

TOLERANCE

Do you often write the fail silent codes like below?

```
try:
    # do what ever you need...
    return "foo"
except:
    # fail silently
    return ""
```

This kind of codes are often found in Django projects or programs which should not raise any exceptions in product mode.

tolerance is a function decorator to make a tolerant function; a function which does not raise any exceptions even there are exceptions. This concept is quite useful for making stable product or `prefer_int` types of code described in Usage section.

1.1 Features

- Convert a function to a tolerant function
- The decorated function returns `substitute` (Default is `None`) when it is not callable. The function returns a "returned value" from `substitute` function when it is callable.
- Ignoreing exceptions can be specified as a exception class list with `exceptions` argument.
- When `fail_silently=False` is passed to the decorated function, the function does not ignore exceptions (the argument name can be changed with making switch function via `argument_switch_generator` function).

1.2 Installation

Use pip like:

```
$ pip install tolerance
```

1.3 Usage

Assume that you need a function which convert a string to an integer when it is possible. Without tolerance, you need to write a code like below

```
>>> # without tolerance
>>> def prefer_int_withot_tolerance(x):
...     try:
...         return int(x)
...     except:
...         # fail silently
...         return x
>>> prefer_int_withot_tolerance(0)
0
>>> prefer_int_withot_tolerance('0')
0
>>> prefer_int_withot_tolerance('zero')
'zero'
```

However, with tolerance, you just need to write a single line code like

```
>>> from tolerance import tolerate
>>> prefer_int = tolerate(lambda x: x)(int)
>>> prefer_int(0)
0
>>> prefer_int('0')
0
>>> prefer_int('zero')
'zero'
```

Or you can use `tolerate` as a function decorator described in PEP-318

```
>>> from tolerance import tolerate
>>> @tolerate(lambda x: x)
... def prefer_int_318(x):
...     return int(x)
>>> prefer_int_318(0)
0
>>> prefer_int_318('0')
0
>>> prefer_int_318('zero')
'zero'
```

The example codes above specify `substitute` argument of `tolerate` function to specify the returning value when the function has failed (`lambda x: x` part). `tolerate` function takes several arguments to configure the function behavior. These arguments are explained in Case study and detailed in API documentation.

1.4 Case study

1.4.1 Q. How can I return the default value when the function fail?

1. Use `substitute` argument to specify the default value like

```
>>> from tolerance import tolerate
>>> @tolerate(substitute='foo')
... def raise_exception():
...     raise Exception
>>> raise_exception()
'foo'
```

1.4.2 Q. How can I change the default value depends on passed arguments?

1. Specify `substitute` argument as a function

```
>>> from tolerance import tolerate
>>> def substitute_function(*args, **kwargs):
...     # do what ever you need, this example simply return 1st argument
...     return args[0]
>>> @tolerate(substitute=substitute_function)
... def raise_exception(*args):
...     raise Exception
>>> raise_exception('bar', 'hoge')
'bar'
```

1.4.3 Q. How can I make the function to ignore only several exceptions?

1. Use `exceptions` argument to specify exceptions which will be ignored.

```
>>> from tolerance import tolerate
>>> exceptions_ignored = (
...     AttributeError,
...     ValueError,
... )
>>> @tolerate(exceptions=exceptions_ignored)
... def raise_exception(x):
...     if x == 0:
...         raise AttributeError
...     elif x == 1:
...         raise ValueError
...     else:
...         raise KeyError
>>> raise_exception(0) is None
True
>>> raise_exception(1) is None
True
>>> raise_exception(2)
Traceback (most recent call last):
    ...
KeyError
```

1.4.4 Q. How can I disable ignoreing exceptions in the decorated function?

1. Pass `fail_silently=False` to the decorated function.

```
>>> from tolerance import tolerate
>>> @tolerate()
... def raise_exception():
...     raise KeyError
>>> raise_exception() is None
True
>>> raise_exception(fail_silently=False)
Traceback (most recent call last):
    ...
KeyError
```

You can change the attribute name with specifing new switch function. It will be explained below.

1.4.5 Q. How can I disable ignoreing exceptions globally?

1. Set `tolerate.disabled = True` to disable tolerance globally.

```
>>> from tolerance import tolerate
>>> @tolerate()
... def raise_exception():
...     raise KeyError
>>> raise_exception() is None
True
>>> tolerate.disabled = True
>>> raise_exception()
Traceback (most recent call last):
    ...
KeyError
>>> # rollback
>>> tolerate.disabled = False
```

1.4.6 Q. How can I disable ignoreing exceptions in complex mannar?

1. Use `switch` argument to specify switch function.

```
>>> from tolerance import tolerate
>>> DEBUG = False
>>> def switch_function(*args, **kwargs):
...     # do what ever you need, this sample check kwargs and DEBUG
...     # remove 'fail_silently' attribute and store
...     fail_silently = kwargs.pop('fail_silently', True)
...     if DEBUG or not fail_silently:
...         # do not ignore exceptions. note that kwargs which does not
...         # have 'fail_silently' is returned back.
...         return False, args, kwargs
...     # do ignore exceptions. note that kwargs which does not have
...     # 'fail_silently' is returned back.
...     return True, args, kwargs
>>> @tolerate(switch=switch_function)
... def raise_exception():
...     raise KeyError
>>> raise_exception() is None
True
>>> raise_exception(fail_silently=False)
Traceback (most recent call last):
    ...
KeyError
>>> DEBUG = True
>>> raise_exception()
Traceback (most recent call last):
    ...
KeyError
```

1.4.7 Q. I just want to change the attribute name, making switch function is too complicated

1. Use `argument_switch_generator` to make switch function.

```
>>> from tolerance import tolerate
>>> from tolerance import argument_switch_generator
>>> switch_function = argument_switch_generator('quiet')
>>> @tolerate(switch=switch_function)
... def raise_exception():
...     raise KeyError
>>> raise_exception() is None
True
>>> # you can use 'quiet=False' instead of 'fail_silently'
>>> raise_exception(quiet=False)
Traceback (most recent call last):
    ...
KeyError
>>> # raise_exception does not know fail_silently so ignore
>>> raise_exception(fail_silently=False) is None
True
```

1.4.8 Q. I want to make the function ignoreing exceptions only when `fail_silently=True` is passed

1. Use `default` argument to tell `argument_switch_generator` function

```
>>> from tolerance import tolerate
>>> from tolerance import argument_switch_generator
>>> switch_function = argument_switch_generator('fail_silently', default=False)
>>> @tolerate(switch=switch_function)
... def raise_exception():
...     raise KeyError
>>> raise_exception() is None
Traceback (most recent call last):
    ...
KeyError
>>> raise_exception(fail_silently=True) is None
True
```

1.4.9 Q. I want to disable the ignoreing exceptions when `verbose=False` is passed

1. Use `reverse` argument to tell `argument_switch_generator` function

```
>>> from tolerance import tolerate
>>> from tolerance import argument_switch_generator
>>> switch_function = argument_switch_generator('verbose', reverse=True)
>>> @tolerate(switch=switch_function)
... def raise_exception():
...     raise KeyError
>>> raise_exception() is None
True
>>> raise_exception(verbose=True)
Traceback (most recent call last):
    ...
KeyError
```

1.4.10 Q. I want to use `fail_silently` argument even in decorated function

1. Use `keep` argument to tell `argument_switch_generator` function

```
>>> from tolerance import tolerate
>>> from tolerance import argument_switch_generator
>>> switch_function = argument_switch_generator('fail_silently', keep=True)
>>> @tolerate(switch=switch_function)
... def raise_exception(**kwargs):
...     if 'fail_silently' in kwargs:
...         raise KeyError
...     return 'Failed!'
>>> raise_exception(fail_silently=True) is None
True
>>> raise_exception(fail_silently=False)
Traceback (most recent call last):
    ...
KeyError
```

CHAPTER

TWO

HOW TO RUN THE TESTS

1. Install requirement packages with `requirements-test.txt`:

   ```
   $ pip install -r requirements-test
   ```

2. Run tests with `nosetests` command provided by nose (it is automatically installed via `requirements-test.txt`)

   ```
   $ nosetests
   ```

All configuration for running tests can be found at `[nosetests]` section of `setup.cfg` file.

CHAPTER

THREE

API DOCUMENTS

3.1 tolerance Package

3.1.1 `tolerance` Package

tolerance

tolerance is a function decorator to make a tolerant function; a function which does not raise any exceptions even there are exceptions. This concept is quite useful for making stable product or `prefer_int` types of code described in Usage section.

`tolerance.`**`tolerate`**(*substitute=None*, *exceptions=None*, *switch=<function switch_function at 0x2b0f050>*)

A function decorator which makes a function fail silently

To disable fail silently in a decorated function, specify `fail_silently=False`. To disable fail silenlty in decorated functions globally, specify `tolerate.disabled`.

Parameters **fn** : function

A function which will be decorated.

substitute : function or returning value

A function used instead of `fn` or returning value when `fn` failed.

exceptions : list of exceptions or None

A list of exception classes or None. If exceptions is specified, ignore exceptions only listed in this parameter and raise exception if the exception is not listed.

switch : function or None

A switch function which determine whether silent the function failar. The function receive `*args` and `**kwargs` which will specified to `fn` and should return status (bool), args, and kwargs. If the function return `False` then agggressive decorated function worked as normal function (raise exception when there is exception). Default switch function is generated by `argument_switch_generator()` with `argument_switch_generator('fail_silently')` so if `fail_silently=False` is specified to the function, the function works as noramlly.

Returns **function** :

A decorated function

Examples

```
>>> #
>>> # use tolerate as a function wrapper
>>> #
>>> parse_int = tolerate()(int)
>>> parse_int(0)
0
>>> parse_int("0")
0
>>> parse_int("zero") is None
True
>>> #
>>> # use tolerate as a function decorator (PIP-318)
>>> #
>>> @tolerate(lambda x: x)
... def prefer_int(x):
...     return int(x)
>>> prefer_int(0)
0
>>> prefer_int("0")
0
>>> prefer_int("zero")
'zero'
>>> #
>>> # filter exceptions be ignored
>>> #
>>> @tolerate(exceptions=(KeyError, ValueError))
... def force_int(x):
...     string_numbers = {
...         'zero': 0,
...         'one': 1,
...         'two': 2,
...         'three': 3,
...         'four': 4,
...         'five': 5,
...         'six': 6,
...         'seven': 7,
...         'eight': 8,
...         'nine': 9
...     }
...     if isinstance(x, (int, float)):
...         return int(x)
...     elif isinstance(x, str):
...         if x in string_numbers:
...             return string_numbers[x]
...         elif x in ('ten', 'hundred', 'thousand'):
...             raise KeyError
...         raise ValueError
...     else:
...         raise AttributeError
>>> force_int('zero')
0
>>> force_int('ten') is None    # KeyError
True
>>> force_int('foo') is None    # ValueError
True
>>> force_int(object)           # AttributeError
```

```
Traceback (most recent call last):
    ...
AttributeError
>>> #
>>> # disable tolerance by passing 'fail_silently=False'
>>> #
>>> force_int('ten', fail_silently=False)    # KeyError
Traceback (most recent call last):
    ...
KeyError
>>> #
>>> # disable tolerance globally by setting 'tolerate.disabled=True'
>>> #
>>> tolerate.disabled = True
>>> force_int('foo')     # ValueError
Traceback (most recent call last):
    ...
ValueError
>>> tolerate.disabled = False    # rollback
```

tolerance.**argument_switch_generator**(*argument_name*, *default=True*, *reverse=False*, *keep=False*)

Create switch function which return the status from specified named argument

Parameters **argument_name** : string

An argument name which is used to judge the status

default : boolean

A default value of this switch function. It is used when specifid `**kwargs` does not have named argument

reverse : boolean

Reverse the status (Default: `False`)

keep : boolean

If it is `True`, keep named argument in `**kwargs`.

Returns **function** :

A switch function which return status, args, and kwargs respectively.

Examples

```
>>> #
>>> # generate switch function with default parameters
>>> #
>>> fn = argument_switch_generator('fail_silently')
>>> # return 'default' value and specified *args and **kwargs when
>>> # 'fail_silently' is not specified in **kwargs
>>> fn() == (True, tuple(), {})
True
>>> # return 'fail_silently' value when it is specified
>>> fn(fail_silently=True) == (True, tuple(), {})
True
>>> fn(fail_silently=False) == (False, tuple(), {})
True
```

```
>>> #
>>> # generate switch function with 'default=False'
>>> #
>>> fn = argument_switch_generator('fail_silently', default=False)
>>> # return 'default' value so 'False' is returned back
>>> fn() == (False, tuple(), {})
True
>>> #
>>> # generate switch function with 'reverse=True'
>>> #
>>> fn = argument_switch_generator('fail_silently', reverse=True)
>>> # 'default' value is independent from 'reverse=True'
>>> fn() == (True, tuple(), {})
True
>>> # 'fail_silently' value is influenced by 'reverse=True'
>>> fn(fail_silently=True) == (False, tuple(), {})
True
>>> fn(fail_silently=False) == (True, tuple(), {})
True
>>> #
>>> # generate switch function with 'keep=True'
>>> #
>>> fn = argument_switch_generator('fail_silently', keep=True)
>>> # 'fail_silently' attribute remains even in returned back kwargs
>>> status, args, kwargs = fn(fail_silently=True)
>>> 'fail_silently' in kwargs
True
```

3.1.2 `decorators` Module

tolerance decorator module

`tolerance.decorators.`**`DEFAULT_TOLERATE_SWITCH`**(**args*, ***kwargs*)
Default tolerate switch function

`tolerance.decorators.`**`tolerate`**(*substitute=None*, *exceptions=None*, *switch=<function switch_function at 0x2b0f050>*)
A function decorator which makes a function fail silently

To disable fail silently in a decorated function, specify `fail_silently=False`. To disable fail silenlty in decorated functions globally, specify `tolerate.disabled`.

Parameters **fn** : function

A function which will be decorated.

substitute : function or returning value

A function used instead of `fn` or returning value when `fn` failed.

exceptions : list of exceptions or None

A list of exception classes or None. If exceptions is specified, ignore exceptions only listed in this parameter and raise exception if the exception is not listed.

switch : function or None

A switch function which determine whether silent the function failar. The function receive `*args` and `**kwargs` which will specified to `fn` and should return status (bool), args, and kwargs. If the function return `False` then agggressive decorated

function worked as normal function (raise exception when there is exception). Default switch function is generated by `argument_switch_generator()` with `argument_switch_generator('fail_silently')` so if `fail_silently=False` is specified to the function, the function works as noramlly.

Returns **function** :

A decorated function

Examples

```
>>> #
>>> # use tolerate as a function wrapper
>>> #
>>> parse_int = tolerate()(int)
>>> parse_int(0)
0
>>> parse_int("0")
0
>>> parse_int("zero") is None
True
>>> #
>>> # use tolerate as a function decorator (PIP-318)
>>> #
>>> @tolerate(lambda x: x)
... def prefer_int(x):
...     return int(x)
>>> prefer_int(0)
0
>>> prefer_int("0")
0
>>> prefer_int("zero")
'zero'
>>> #
>>> # filter exceptions be ignored
>>> #
>>> @tolerate(exceptions=(KeyError, ValueError))
... def force_int(x):
...     string_numbers = {
...         'zero': 0,
...         'one': 1,
...         'two': 2,
...         'three': 3,
...         'four': 4,
...         'five': 5,
...         'six': 6,
...         'seven': 7,
...         'eight': 8,
...         'nine': 9
...     }
...     if isinstance(x, (int, float)):
...         return int(x)
...     elif isinstance(x, str):
...         if x in string_numbers:
...             return string_numbers[x]
...         elif x in ('ten', 'hundred', 'thousand'):
...             raise KeyError
```

```
...         raise ValueError
...     else:
...         raise AttributeError
>>> force_int('zero')
0
>>> force_int('ten') is None    # KeyError
True
>>> force_int('foo') is None    # ValueError
True
>>> force_int(object)           # AttributeError
Traceback (most recent call last):
    ...
AttributeError
>>> #
>>> # disable tolerance by passing 'fail_silently=False'
>>> #
>>> force_int('ten', fail_silently=False)   # KeyError
Traceback (most recent call last):
    ...
KeyError
>>> #
>>> # disable tolerance globally by setting 'tolerate.disabled=True'
>>> #
>>> tolerate.disabled = True
>>> force_int('foo')    # ValueError
Traceback (most recent call last):
    ...
ValueError
>>> tolerate.disabled = False   # rollback
```

3.1.3 `functional` Module

3.1.4 `utils` Module

tolerance utility module

`tolerance.utils.`**`argument_switch_generator`**(*argument_name*, *default=True*, *reverse=False*, *keep=False*)

Create switch function which return the status from specified named argument

Parameters **argument_name** : string

An argument name which is used to judge the status

default : boolean

A default value of this switch function. It is used when specifid `**kwargs` does not have named argument

reverse : boolean

Reverse the status (Default: `False`)

keep : boolean

If it is `True`, keep named argument in `**kwargs`.

Returns **function** :

A switch function which return status, args, and kwargs respectively.

Examples

```
>>> #
>>> # generate switch function with default parameters
>>> #
>>> fn = argument_switch_generator('fail_silently')
>>> # return `default` value and specified *args and **kwargs when
>>> # `fail_silently` is not specified in **kwargs
>>> fn() == (True, tuple(), {})
True
>>> # return `fail_silently` value when it is specified
>>> fn(fail_silently=True) == (True, tuple(), {})
True
>>> fn(fail_silently=False) == (False, tuple(), {})
True
>>> #
>>> # generate switch function with `default=False`
>>> #
>>> fn = argument_switch_generator('fail_silently', default=False)
>>> # return `default` value so `False` is returned back
>>> fn() == (False, tuple(), {})
True
>>> #
>>> # generate switch function with `reverse=True`
>>> #
>>> fn = argument_switch_generator('fail_silently', reverse=True)
>>> # `default` value is independent from `reverse=True`
>>> fn() == (True, tuple(), {})
True
>>> # `fail_silently` value is influenced by `reverse=True`
>>> fn(fail_silently=True) == (False, tuple(), {})
True
>>> fn(fail_silently=False) == (True, tuple(), {})
True
>>> #
>>> # generate switch function with `keep=True`
>>> #
>>> fn = argument_switch_generator('fail_silently', keep=True)
>>> # `fail_silently` attribute remains even in returned back kwargs
>>> status, args, kwargs = fn(fail_silently=True)
>>> 'fail_silently' in kwargs
True
```

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

t

PYTHON MODULE INDEX

t

INDEX