pydecorator Documentation

Release 1.0

Lucian Cooper

Jan 18, 2019

Documentation

This package contains decorators that facilitate different types of tasks

Current version: 1.0
Github
PyPi

CHAPTER 1

Installation

Use `pip` via PyPi:

```
pip install pydecorator
```

Or use `git`:

```
git clone git://github.com/luciancooper/pydecorator.git pydecorator
cd pydecorator
python setup.py install
```

1.1 Generators

Contents

All functions decorate either a generator function or a function that yields an iterable. They return a specified type like `list` or `dict`. All of the decorators in this category are *signature preserving*, therefore they can be used to decorate class methods as well as standalone functions.

1.1.1 `list`

This function decorates either a generator function or a function that yields an iterable. For example:

```
@pydecorator.list
def generate_list():
    """Generates a list"""
    for i in "generator":
        yield i
```

The `generate_list` function will return a list, like so:

```
>>> generate_list()
['g', 'e', 'n', 'e', 'r', 'a', 't', 'o', 'r']
```

1.1.2 tuple

This is the same as the `pydecorator.list` function, but for tuples. For example:

```
@pydecorator.tuple
def generate_tuple():
    """Generates a tuple"""
    for i in "generator":
        yield i
```

The `generate_tuple` function will return a tuple, like so:

```
>>> generate_tuple()
('g', 'e', 'n', 'e', 'r', 'a', 't', 'o', 'r')
```

1.1.3 set

This is the same as the `pydecorator.set` function, but for sets. For example:

```
@pydecorator.set
def generate_set():
    """Generates a set"""
    for i in "ababc":
        yield i
```

The `generate_set` function will return a set, like so:

```
>>> generate_set()
{'a', 'c', 'b'}
```

1.1.4 dict

This must decorate a generator function that yields key value pairs, or a function that returns an iterable of key value tuple pairs. For example:

```
@pydecorator.dict
def keymap():
    """Generates a dict"""
    for k,v in zip('ABC','XYZ'):
        yield k,v
```

The `keymap` function will return a dict:

```
>>> keymap()
{'A': 'X', 'B': 'Y', 'C': 'Z'}
```

1.1.5 str

This can decorate either a generator function that yields strings, or a function that returns an iterable of strings. For example:

```
@pydecorator.str
def word():
    """Generates a string"""
    for letter in ['g','e','n','e','r','a','t','e']:
        yield letter
```

The word function will return a concatenated string:

```
>>> word()
'generator'
```

1.2 Transforms

Contents

These functions transform the output of the decorated function. They are *signature preserving*, so can therefore be used to decorate class methods as well as standalone functions

1.2.1 transpose

This function can decorate either a generator function that yields iterables, or a function that returns an iterable containing iterable values. For example:

```
@pydecorator.transpose
def matrix():
    """Generates 4 lists of 3 values"""
    for i in range(4):
        yield ["%i-%i"%(i,j) for j in range(3)]
```

The matrix function will yield 3 lists of 4 values, like so:

```
>>> [*matrix()]
[['0-0', '1-0', '2-0', '3-0'], ['0-1', '1-1', '2-1', '3-1'], ['0-2', '1-2', '2-2', '3-
↪2']]
```

1.2.2 list_transpose

This function does the same thing as *transpose*, but the decorated function will return a list, instead of being a generator. For example:

```
@pydecorator.list_transpose
def matrix():
    """Generates 4 lists of 3 values"""
    for i in range(4):
        yield ["%i-%i"%(i,j) for j in range(3)]
```

The `matrix` function will return a list containing 3 lists of 4 values each:

```
>>> matrix()
[['0-0', '1-0', '2-0', '3-0'], ['0-1', '1-1', '2-1', '3-1'], ['0-2', '1-2', '2-2', '3-
↪2']]
```

1.3 Files

Contents

The file functions are *signature changing* decorators. They allow you to read and write files by write functions that handle only a single line at a time.

1.3.1 file_reader

This decorates a function that accepts a string as the first argument. It changes the signature of the decorated function to `(filepath,*args,**kwargs)`. The returned function is a generator that will open the file specified in the `filepath` argument, and then yield the result of calling the decorated function with the signature `(line,*args,**kwargs)` for each line in that file. For example:

```
@pydecorator.file_reader
def read_csv(line):
    return line.strip().split(",")
```

Given the file `sometext.csv` which contains:

```
A,B,C
D,E,F
G,H,I
```

The `read_csv` function will return:

```
>>> read_csv('sometext.csv')
[['A', 'B', 'C'], ['D', 'E', 'F'], ['G', 'H', 'I']]
```

1.3.2 file_writer

This decorates a generator function that yields each line in a file. It changes the signature of the decorated function to `(filepath,*args,**kwargs)`. The returned function calls the original function with the signature *(*args,**kwargs)*, and writes out each line to the file specified by `filepath`. For example:

```
@pydecorator.file_writer
def write_csv():
    for line in ["ABC","DEF","GHI"]:
        yield line
```

Calling `write_csv('newfile.txt)` would create a `newfile.txt` with the contents:

```
ABC
DEF
GHI
```

1.4 Sorting

Contents

All the decorators in this category take a single keyword argument `duplicate_values`, a boolean value that indicates whether or not the sorting algorithm removes extra duplicate values. The default is `True`, meaning duplicate values are allowed, and not removed.

1.4.1 Comparison function decorators

The following three decorators, *mergesort*, *mergesort_map*, and *mergesort_index* are all used to decorate a comparison function that follows the signature stipulated below.

The decorated comparator must have the signature `(a,b)`, and must return either -1, 0, or 1, indicating the following:

Returned	Interpretation
0	a = b
-1	a < b
1	a > b

These decorators change the signature of the function they decorate, therefore if they are used to decorate a class method, `@staticmethod` must be stacked on top like this:

```
class Sorter():

    @staticmethod
    @pydecorator.mergesort(duplicate_values=True)
    def sort(a,b):
        return 1 if a > b else -1 if a < b else 0
```

mergesort

The decorated function will have the signature `(array)`, where `array` is either a generator or an indexable collection. The function will return the input as a sorted list:

```
@pydecorator.mergesort(duplicate_values=False)
def sorted_set(a,b):
    return 1 if a > b else -1 if a < b else 0
```

```
>>> sorted_set([1,3,0,1,2])
[0, 1, 2, 3]
```

mergesort_map

The decorated function will have the signature `(array)`, where `array` is either a generator or an indexable collection. The function will return a list of integers, each within the range (`0 ,..., len(array)-1`). Each int in the returned array of the returned each corresponding to an index the input as a sorted list:

```
@pydecorator.mergesort_map(duplicate_values=True)
def sorted_map(a,b):
    return 1 if a > b else -1 if a < b else 0
```

```
>>> test = ['b','d','a','b','c']
>>> test
['b', 'd', 'a', 'b', 'c']
>>> index = sorted_map(test)
>>> index
[2, 0, 3, 4, 1]
>>> [test[i] for i in index]
['a', 'b', 'b', 'c', 'd']
```

mergesort_index

The decorated function will have the signature `(index,array)`, where `array` is either a generator or an indexable collection, and `index` is an indexable collection of integers, each within the range (`0 ,..., len(array)-1`), corresponding to an element in `array`. The function will return the input `index`, in sorted order (when mapped to `array`). A simple example:

```
@pydecorator.mergesort_index(duplicate_values=False)
def sorted_indexes(a,b):
    return 1 if a > b else -1 if a < b else 0
```

```
>>> test = ['b','d','a','b','c']
>>> test
['b', 'd', 'a', 'b', 'c']
>>> index = sorted_indexes(range(len(test)),test)
>>> index
[2, 0, 4, 1]
>>> [test[i] for i in index]
['a', 'b', 'c', 'd']
```

Note: The `mergesort_index` decorator is meant to be used to create functions that are part of more complex routines, such as sorting multi-dimensional data, and does not have much of a practical application as a standalone function. Use `mergesort_map` instead.

1.4.2 sorted

This function can decorate either a generator function or a function that returns an indexable collection. It preserves the decorated functions signature. Here's a simple example:

```
@pydecorator.sorted(duplicate_values=False)
def make_list():
    return [1,0,3,5,1]
```

```
>>> make_list()
[0, 1, 3, 5]
```

1.5 Numpy

Contents

These functions are meant to decorate generator functions to return *numpy* arrays. They are *signature preserving* decorators, and can therefore be used to decorate class methods as well as standalone functions.

1.5.1 np_c

This decorates generator functions and applys the numpy.c_ routine to the output. For example:

```
@pydecorator.np_c
def array():
    for col in [[1,2,3],[4,5,6],[7,8,9]]:
        yield col
```

```
>>> array()
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

1.5.2 np_r

This decorates generator functions and applys the numpy.r_ routine to the output. For example:

```
@pydecorator.np_r
def array():
    for row in [[1,2,3],[4,5,6],[7,8,9]]:
        yield row
```

```
>>> array()
[1 2 3 4 5 6 7 8 9]
```

1.5.3 np_rows

This decorates generator functions and returns their output as the rows of a numpy.array. For example:

```
@pydecorator.np_rows
def array():
    for row in [[1,2,3],[4,5,6],[7,8,9]]:
        yield row
```

```
>>> array()
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

1.6 Pandas

Contents

These functions decorate are used to decorate generators, returning various pandas classes. They are *signature preserving* decorators, and can therefore be used to decorate class methods as well as standalone functions.

1.6.1 pd_dfrows

Decorates a generator function that yields items in the form `[row,...]`. The resulting function returns a pandas dataframe. Has the optional keyword argument `columns`, which allows you to name the columns of the returned dataframe. For example:

```
@pydecorator.pd_dfrows(columns=['a','b'])
def dataframe():
    for row in [[0,1],[2,3],[4,5]]:
        yield row
```

```
>>> dataframe()
a  b
0  0  1
1  2  3
2  4  5
```

1.6.2 pd_dataframe

Decorates a generator function that yields items in the form `(key,[row,..])`, and returns a pandas dataframe. Has the optional keyword arguments `index` and `columns`, which allow you to name the index and columns of the returned dataframe. For example:

```
@pydecorator.pd_dataframe(index='i',columns=['a','b'])
def dataframe():
    for k,v in zip('xyz',[[0,1],[2,3],[4,5]]):
        yield k,v
```

```
>>> dataframe()
   a  b
i
x  0  1
y  2  3
z  4  5
```

1.6.3 pd_multiframe

Decorates a generator function that yields items in the form `([key,..],[row,..])`, and returns a pandas dataframe with a MultiIndex. Has the optional keyword arguments `index` and `columns`, which allow you to name the index and columns of the returned dataframe. For example:

```
@pydecorator.pd_multiframe(index=['i','j'],columns=['a','b'])
def dataframe():
    for k1,k2,v in zip('xyz','abc',[[0,1],[2,3],[4,5]]):
        yield (k1,k2),v
```

```
>>> dataframe()
     a  b
i j
x a  0  1
y b  2  3
z c  4  5
```

1.6.4 pd_series

Decorates a generator function that yields items in the form `(key,value)`, and returns a pandas series. Has the optional keyword arguments `index` and `name`, which allow you to name the index of the series, and the series itself, respectively. For example:

```
@pydecorator.pd_series(index='i',name='a')
def series():
    for k,v in zip('xyz',[0,1,2]):
        yield k,v
```

```
>>> series()
i
x    0
y    1
z    2
Name: a, dtype: int64
```

1.6.5 pd_multiseries

Decorates a generator function that yields items in the form `([key,..],value)`, and returns a pandas series with a MultiIndex. Has the optional keyword arguments `index` and `name`, which allow you to provide names for the index of the series and the name of the series itself, respectively. For example:

```
@pydecorator.pd_multiseries(index=['i','j'],name='a')
def series():
    for k1,k2,v in zip('xyz','abc',[0,1,2]):
        yield (k1,k2),v
```

```
>>> series()
i  j
x  a    0
y  b    1
z  c    2
Name: a, dtype: int64
```

1.6.6 pd_index

Decorates a generator function that yields non iterable items, and returns a pandas Index. Has the optional keyword argument `name`, which allows you to provide a name for the returned index. For example:

```
@pydecorator.pd_index(name='i')
def index():
    for k in 'xyz':
        yield k
```

```
>>> index()
Index(['x', 'y', 'z'], dtype='object', name='i')
```

1.6.7 pd_multi_index

Decorates a generator function that yields items in the form `[key,...]`, and returns a pandas MultiIndex. Has the optional keyword argument `names`, which allows you to provide the level names for the returned index. For example:

```
@pydecorator.pd_multi_index(names=['i','j'])
def index():
    for k1,k2 in zip('xyz','abc'):
        yield (k1,k2)
```

```
>>> index()
MultiIndex(levels=[['x', 'y', 'z'], ['a', 'b', 'c']],
           labels=[[0, 1, 2], [0, 1, 2]],
           names=['i', 'j'])
```