argskwargs

Release

Jun 22, 2017

Contents

`argskwargs` is a small Python library that provides a flexible container for positional and keyword arguments.

CHAPTER 1

Installation

```
pip install argskwargs
```

`argskwargs` can be used on Python 3.3+ and Python 2.6+.

CHAPTER 2

Usage

argskwargs provides a small container class to hold arbitrary positional arguments (args) and keyword arguments (kwargs). This container is essentially the same as a (args, kwargs) tuple, but with a nice and small API on top to keep your code simple and clear.

Passing around arguments for a function without actually calling that function (at least not yet) typically involves two variables that are closely kept together:

- a tuple, often called args
- a dict, often called kwargs

argskwargs simplifies this clunky and error-prone code pattern by putting these two values inside a small container:

```
>>> from argskwargs import argskwargs
>>> my_args = argskwargs(1, 2, foo='bar')
>>> my_args
argskwargs(1, 2, foo='bar')
```

An argskwargs container simply stores args and kwargs. You can unpack the container to obtain a tuple and a dict again:

```
>>> x, y = my_args
>>> x
(1, 2)
>>> y
{'foo': 'bar'}
```

Alternatively, access the .args and .kwargs attributes:

```
>>> my_args.args
(1, 2)
>>> my_args.kwargs
{'foo': 'bar'}
```

Let's define a function that simply prints out anything that is passed to it:

```
>>> import pprint
>>> def print_arguments(*args, **kwargs):
...     print('positional arguments ' + str(args))
...     print('keyword arguments ' + pprint.pformat(kwargs))
```

(Note: dictionary order cannot be relied on in most Python versions. The use of `pformat()` makes the output deterministic, since that function will sort the dict keys. All the sample code in this documentation is actually executed as part of the tests for this library, and deterministic output is required for those doctests to pass successfully.)

This function can be called directly using 'splat' syntax:

```
>>> print_arguments(*my_args.args, **my_args.kwargs)
positional arguments (1, 2)
keyword arguments {'foo': 'bar'}
```

Arguably, this is not much better than using using two variables to for the positional and keyword arguments.

Let's see what makes `argskwargs` useful.

Here is another way to do the same using the `.apply()` method:

```
>>> my_args.apply(print_arguments)
positional arguments (1, 2)
keyword arguments {'foo': 'bar'}
```

Since this is the typical use case for `argskwargs`, you can also omit the `.apply()` and call the instance directly:

```
>>> my_args(print_arguments)
positional arguments (1, 2)
keyword arguments {'foo': 'bar'}
```

As you can see the code is inverted: the callable is passed to the arguments, instead of the other way around, as is usually the case.

Now, assume that you want to pass more arguments to `print_arguments` than those stored in the `argskwargs` instance. Just pass them in:

```
>>> my_args(print_arguments, 'another', oh='yes')
positional arguments (1, 2, 'another')
keyword arguments {'foo': 'bar', 'oh': 'yes'}
```

The additional positional arguments extend the existing positional arguments, and the additional keyword arguments augment (or override) the existing positional arguments.

If you just want to extend the arguments without calling a function, use the `.copy()` method, which does exactly that:

```
>>> more_args = my_args.copy(3, 4, abc='xyz')
>>> more_args
argskwargs(1, 2, 3, 4, abc='xyz', foo='bar')
```

This new argument container can now be used like the original one:

```
>>> more_args(print_arguments)
positional arguments (1, 2, 3, 4)
keyword arguments {'abc': 'xyz', 'foo': 'bar'}
```

In a sense, `argskwargs` is the missing companion to `functools.partial()` from the Python standard library. A partial function (or 'partial object') can also hold positional and keyword arguments, but cannot be used without a callable.

An `argskwargs` container can create a partial function by providing a callable to its `.partial()` method:

```
>>> f = my_args.partial(print_arguments)
```

The resulting partial function can be called as usual:

```
>>> f()
positional arguments (1, 2)
keyword arguments {'foo': 'bar'}
```

And of course additional arguments can be provided:

```
>>> f(3, 4, abc='xyz')
positional arguments (1, 2, 3, 4)
keyword arguments {'abc': 'xyz', 'foo': 'bar'}
```

You can pass more arguments in one go when creating the partial function, and even more when calling it:

```
>>> g = my_args.partial(print_arguments, 3, 4, foo='foofoo')
>>> g(5, 6, bar='baz')
positional arguments (1, 2, 3, 4, 5, 6)
keyword arguments {'bar': 'baz', 'foo': 'foofoo'}
```

You may want to avoid the more complex forms, since those will likely not improve code clarity. Or as the Python mantras go: *readability counts* and *simple is better than complex*.

CHAPTER 3

API

CHAPTER 4

Contributing

The source code and issue tracker for this package can be found on Github:

> https://github.com/wbolster/argskwargs

CHAPTER 5

License

..include:: ../LICENSE.rst