# privates Documentation

***Release 2017.1.1alpha1***

**Dave Willmer**

**Sep 16, 2017**

# Contents:

CHAPTER 1

# Changelog

## 2017.1

### New Features

- `no_mutations` context manager

### Bug Fixes

### Enhancements

CHAPTER 2

# Overview

*privates* is a library which extends the functionality of python by exposing previously hidden/private attributes through a nice, easy API.

This library was created to have a thoroughly tested community resource for the helper functions we sometimes need in our applications, but which involve private/hidden attributes.

## No mutations context

The `no_mutations()` context manager throws a `MutationError` if it detects a mutation on the passed object within the scope of that context:

```
>>> from privates import no_mutations
>>> x = {'a': 1, 'b': 2}
>>>
>>> # This works fine as it's reading
>>> # from the dictionary
>>> with no_mutations(x):
...     y = x['a']
>>>
>>> # This throws a mutation error because
>>> # the dict was modified within the context
>>> with no_mutations(x):
...     x['c'] = 3
Traceback (most recent call last):
    ...
privates.core.errors.MutationError: The dict was mutated
```

This behaviour only applies within the scope of the context manager:

```
>>> from privates import no_mutations
>>> x = {'a': 1, 'b': 2}
>>>
>>> # This works fine because it is
```

```
>>> # outside the scope of the context
>>> # manager.
>>> x['c'] = 3
>>>
>>> # This throws an error
>>> with no_mutations(x):
...     x['d'] = 4
Traceback (most recent call last):
    ...
privates.core.errors.MutationError: The dict was mutated
```

However, even though the error is thrown, the change is still made to the dictionary:

```
>>> from privates import no_mutations
>>> x = {'a': 1, 'b': 2}
>>> with no_mutations(x):
...     x['c'] = 1000
Traceback (most recent call last):
    ...
privates.core.errors.MutationError: The dict was mutated
>>> assert x['c'] == 1000
```

This is so that any calling code which would like to allow the mutation can do:

```
>>> from privates import no_mutations, MutationError
>>> x = {'a': 1, 'b': 2}
>>> def my_func(x):
...     x['c'] = 3
>>> try:
...     my_func(x)
... except MutationError:
...     pass
```

## NamedStruct

The `privates.named_struct.NamedStruct` is a typed struct class which allows inheritance (unlike *tuple/typing.NamedTuple*), and which can be automatically converted to an instance of a *numba.jitclass* using its classmethod *.create(**kwargs)*.

A simple 2D Point and Rectangle class can therefore be defined as follows:

```
>>> import numpy as np
>>> from privates import NamedStruct
>>>
>>> class Point(NamedStruct):
...     x: float
...     y: float
...
...     def distance_from_origin(self):
...         return np.sqrt(self.x**2 + self.y**2)
>>>
>>> class Rectangle(Point):
...     width: float
...     height: float
...
...     def area(self):
```

```
...         return self.width * self.height
>>>
>>> p = Point(1.0, 1.0)
>>> p.distance_from_origin()
1.414...
>>> r = Rectangle(0.0, 0.0, 5.0, 4.0)
>>> r.area()
20.0
```

The attributes and methods from Point are inherited by Rectangle, and the use of *numba.types* as the type declarations allows jitclasses to be used directly without extra decorators:

```
>>> from numba.types import float64
>>> from privates import NamedStruct
>>>
>>> class Point(NamedStruct):
...     x: float64
...     y: float64
...
...     def distance_from_origin(self):
...         return sqrt(self.x**2 + self.y**2)

# >>> # TODO: fix this for v2017.1
# >>> p = Point.create(x=3.0, y=4.0)
# >>> p.distance_from_origin()
# 5.0
```

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search