Pydecorate Documentation

Release 0.4.1.dev32+g9b13c1a

Hrobjartur Thorsteinsson

Apr 02, 2024

CONTENTS

1 Contents **3**
- 1.1 1. Installation . . . 3
  - 1.1.1 1.1. Development Installation . . . 3
- 1.2 Usage . . . 3
  - 1.2.1 features . . . 4
  - 1.2.2 placement . . . 7
  - 1.2.3 styles . . . 10
- 1.3 pydecorate . . . 11
  - 1.3.1 pydecorate package . . . 11

2 Indices and tables **15**

Python Module Index **17**

Index **19**

Pydecorate is a package for decorating PIL images with logos, texts, and color scales.

The source code of the package can be found at on GitHub.

CHAPTER

ONE

CONTENTS

1.1 1. Installation

Pydecorate can be installed in a conda environment by using the conda-forge channel:

```
conda install -c conda-forge pydecorate
```

Or using pip:

```
pip install pydecorate
```

1.1.1 1.1. Development Installation

To install from source, clone the git repository:

```
git clone https://github.com/pytroll/pydecorate.git
```

Then use pip to install the package in development mode:

```
pip install -e .
```

1.2 Usage

When using the decorator the following modules might need to be imported,

```
>>> from PIL import Image
>>> from pydecorate import DecoratorAGG
>>> import aggdraw
```

From the extracted source directory, you can read in a demonstration image to play with,

```
>>> img = Image.open('BMNG_clouds_201109181715_areaT2.png')
```

To begin work on decorating this PIL image object you simply instantiate a decorator object with the PIL image as argument,

```
>>> dc = DecoratorAGG(img)
```

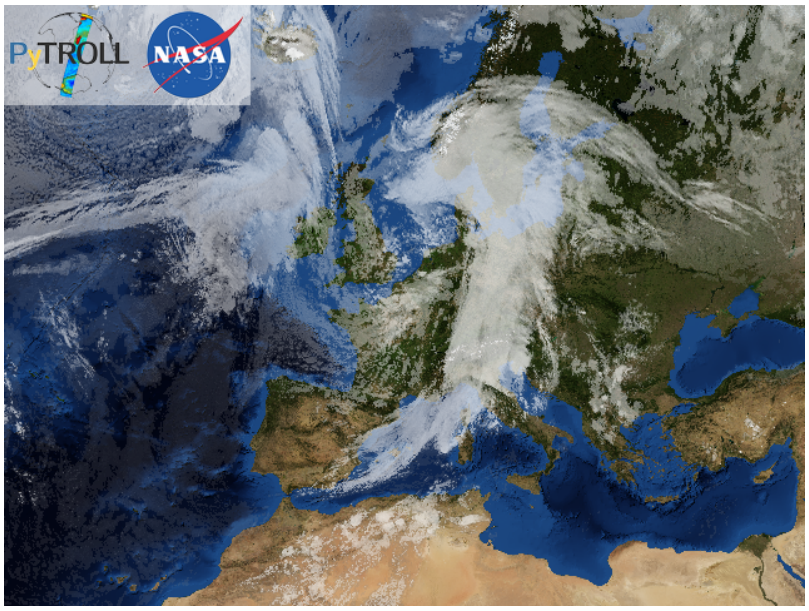
1.2.1 features

Pydecorate supports adding features such as picures and logos, text, color scales and legends to your PIL image.

logos

A simple use case is to add a couple of logos. From the extracted source directory you can add a couple of demonstration logos:

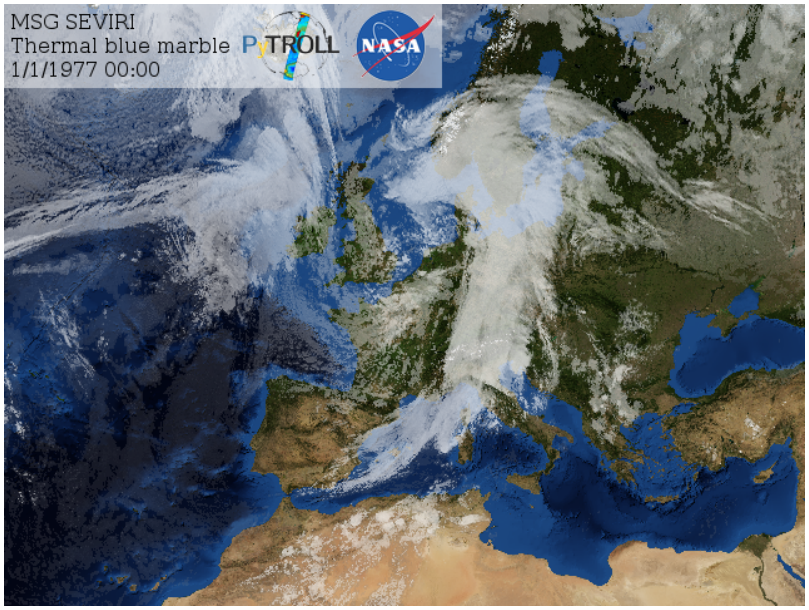
```
>>> dc.add_logo("logos/pytroll_light_big.png",height=80.0)
>>> dc.add_logo("logos/NASA_Logo.gif")
>>>
>>> img.show()
```



text

To add text, you could do:

```
>>> dc.add_text("MSG SEVIRI\nThermal blue marble\n1/1/1977 00:00")
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_logo("logos/NASA_Logo.gif")
>>>
>>> img.show()
```



Notice how the height style of the logos follow the height of the text that was entered. This is because the current height style of the decorator is automatically set to match the space required by the text.

If the add_text operation fails it is may be necessary to specify the full path to your truetype font by loading an aggdraw font object:

```
>>> font=aggdraw.Font("blue","/usr/share/fonts/truetype/ttf-dejavu/DejaVuSerif.ttf",
↪size=16)
```

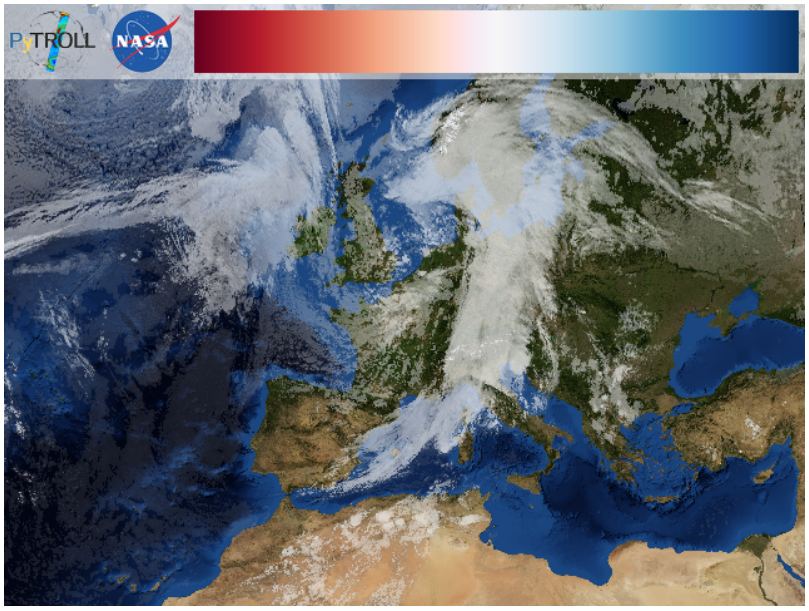
In doing so, you also can also control the font size and font colour. To use this font you must pass it as an optional argument as so,

```
>>> dc.add_text("MSG SEVIRI\nThermal blue marble\n1/1/1977 00:00",font=font)
```

scales and legends (DRAFT)

PyDecorate can work with trollimage colormap objects to add colour scales. To add some logos along with a standard scale feature based on the 'rdbu' scale from trollmap one might do as follows,

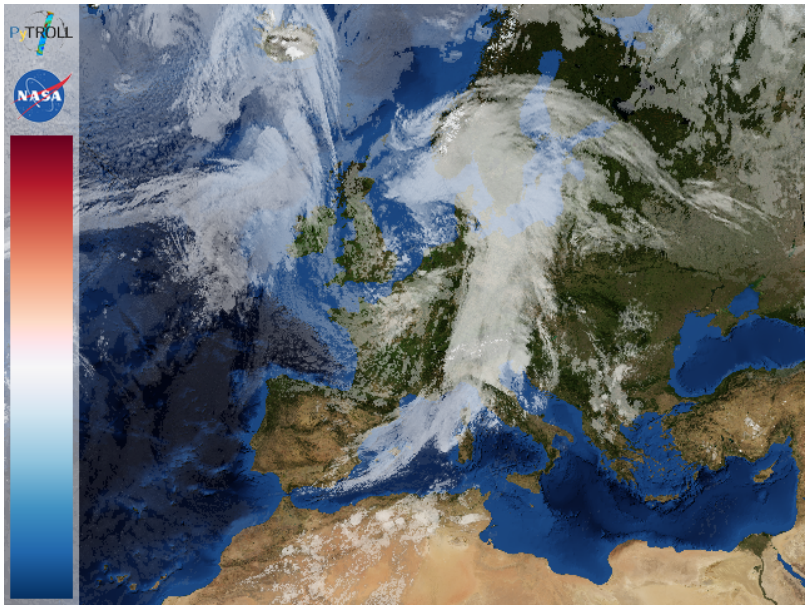
```
>>> from trollimage.colormap import rdbu
>>> rdbu.set_range(-80,30)
>>>
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_logo("logos/NASA_Logo.gif")
>>> dc.add_scale(rdbu,extend=True)
```



Note that the extend=True option sets the scale feature to extend to the full available space. Without this option the scale will inherit the previous width, or one might pass a specific width as argument. See more on this topic in the Styles section below.

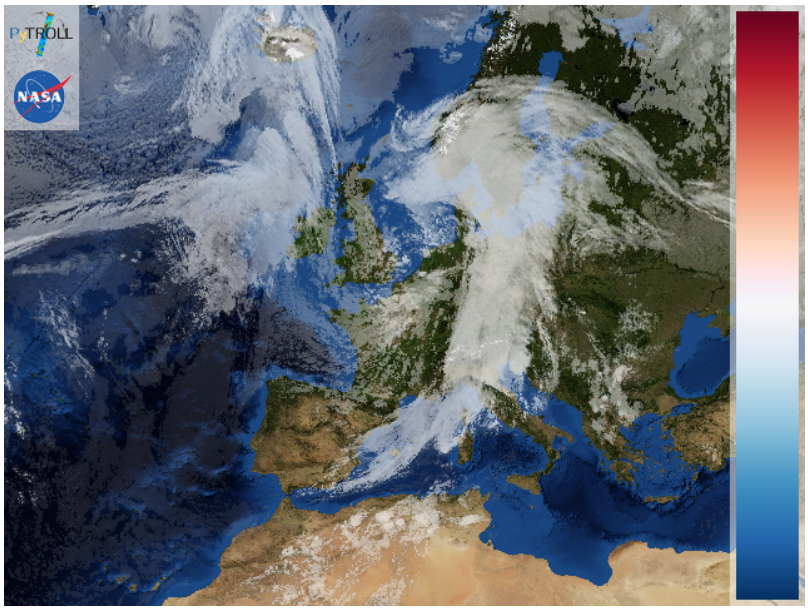
Off course the placement of the logos and scale feature is very flexible. For example if one wanted to arrange the logos vertically followed by a vertical color scale one simply precedes the code with a write_vertically() directive,

```
>>> dc.write_vertically()
>>>
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_logo("logos/NASA_Logo.gif")
>>> dc.add_scale(rdbu,extend=True)
```



To separate the color scale to the right hand side, simply throw in an alignment directive between the logos and the scale,

```
>>> dc.write_vertically()
>>>
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_logo("logos/NASA_Logo.gif")
>>>
>>> dc.align_right()
>>>
>>> dc.add_scale(rdbu,extend=True)
```



More on feature placement options in the following section.

1.2.2 placement

The decorator allows the cursor to be relocated and alligned to different sides of the image. By default the cursor writes horizontally from the top-left corner. The cursor can however be easily relocated at any other side of the image, and the vertical and horizontal write orientation can be changed.

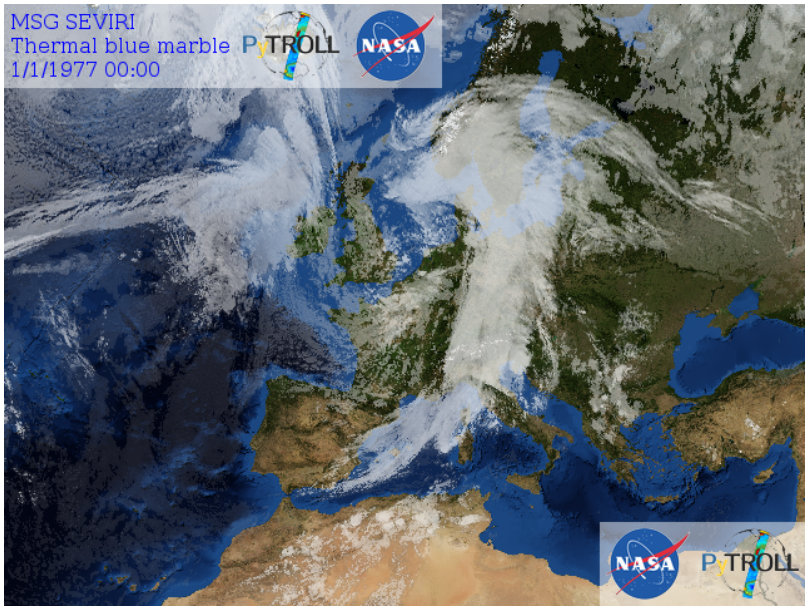
alignment

Continuing from the previous example, we can align the cursor to the bottom-right corner, by executing

```
>>> dc.align_right()
>>> dc.align_bottom()
```

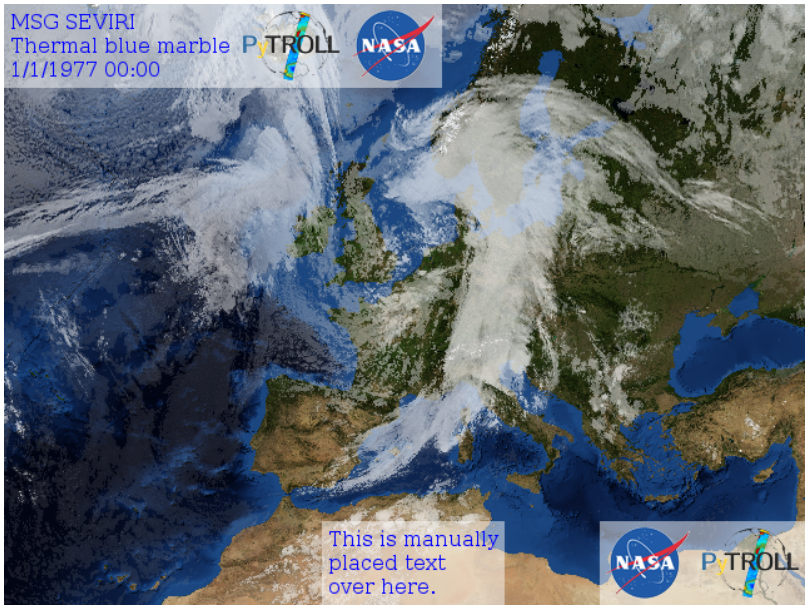
New features will now be written horizontally from the bottom-right corner progressing leftwards. E.g.

```
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_logo("logos/NASA_Logo.gif")
```



Note: Currently the decorator does not provide an easy option for centered placement of features. However the cursor position may be set manually as part of the style arguments to achieve this kind of placement, e.g.

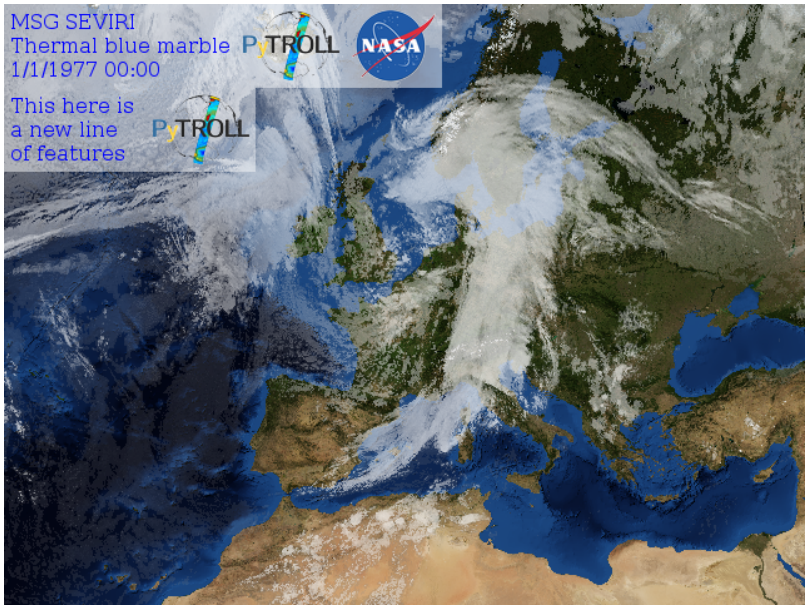
```
>>> dc.add_text("This is a manually\nplaced text\nover here.", cursor=[400,480])
```



new line

As with typewriters, the decorator can also progress to a new line of features. Starting from our first example,

```
>>> dc.new_line()
>>> dc.add_text("This here is\na new line\nof features")
>>> dc.add_logo("logos/pytroll_light_big.png")
```

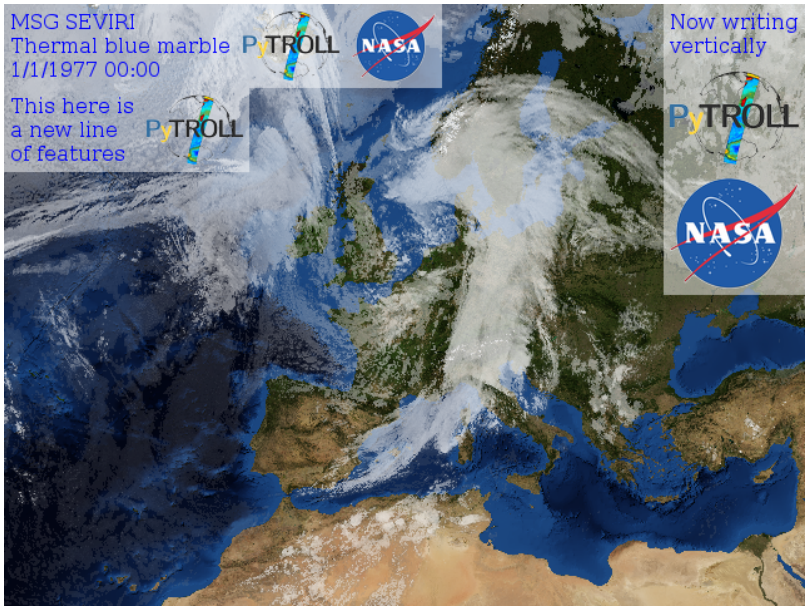


horizontal/vertical writing

The orientation of the cursor writes can be changed from vertical to horizontal writing. The following statements will write some features vertically,

```
>>> dc.align_right()
>>> dc.write_vertically()
>>>
>>> dc.add_text("Now writing\nvertically", height=0)
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_logo("logos/NASA_Logo.gif")
```

Note that resetting the height of text to zero prevents the text feature from inheriting the height of the previously added feature and allows it to expand to the necessary height.



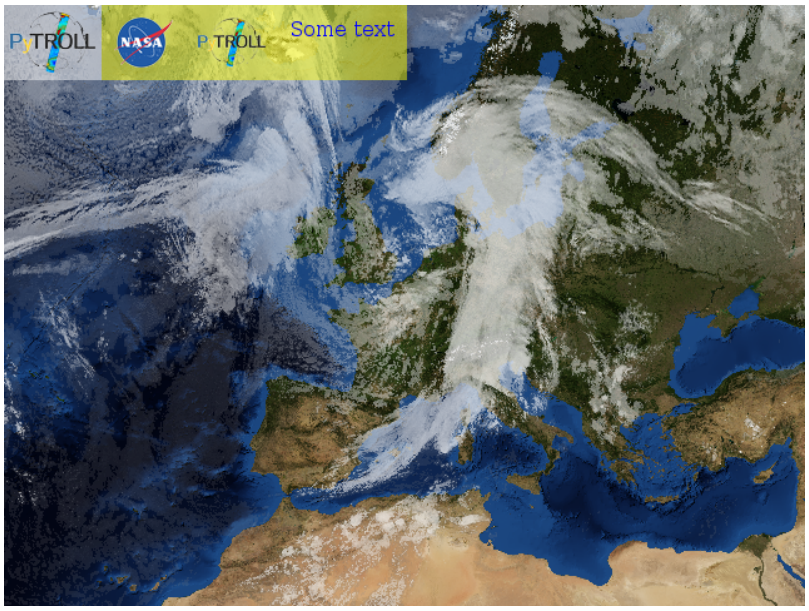
1.2.3 styles

The decorator style settings is a dictionary of options that is retentive between successive operations (state machine), all except for the `cursor` attribute which is propagative as mentioned before,

```
>>> default_style_dict = {
'cursor':[0,0],
'margins':[5,5],
'height':60,
'width':60,
'propagation':[1,0],
'newline_propagation':[0,1],
'alignment':[0.0,0.0],
'bg':'white',
'bg_opacity':127,
'line':"black",
'line_width':1,
'line_opacity':255,
'outline':None,
'outline_width':1,
'outline_opacity':255,
'fill':'black',
'fill_opacity':255,
'font':None,
'start_border':[0,0],
'extend':False,
'tick_marks':1.0,
'minor_tick_marks':0.1
}
```

For example, when if at some point we define a new background color and margins, this style does not have to be repeated on successive calls,

```
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_logo("logos/NASA_Logo.gif",margins=[10,10],bg='yellow')
>>> dc.add_logo("logos/pytroll_light_big.png")
>>> dc.add_text("Some text",font=font)
```



1.3 pydecorate

1.3.1 pydecorate package

Subpackages

pydecorate.fonts package

Module contents

pydecorate.tests package

Submodules

pydecorate.tests.test_decorator_agg module

Tests for the aggdraw-based decorator.

pydecorate.tests.test_decorator_agg.**assert_colorbar_increasing_tick_order**(*draw_text_wrapper*)
→ None

pydecorate.tests.test_decorator_agg.**assert_colorbar_orientation_alignment**(*img_arr: ndarray[Any, dtype[_ScalarType_co]]*, *orientation_func_name: str*, *align_func_name: str*, *clims_flipped: bool*) → None

pydecorate.tests.test_decorator_agg.**test_add_logo**(*tmp_path*)

pydecorate.tests.test_decorator_agg.**test_colorbar**(*tmp_path*, *orientation_func_name*, *align_func_name*, *clims*)

pydecorate.tests.test_legacy module

Basic test cases of pydecorate.

NOTE: These aren't proper unit tests

pydecorate.tests.test_legacy.**test_style_retention**(*tmp_path*)

Module contents

Submodules

pydecorate.decorator module

PIL-based image decoration class.

class pydecorate.decorator.**Decorator**(*image*)

Bases: *DecoratorBase*

PIL-based image decoration class.

add_logo(*logo_path*, ***kwargs*)

add_scale(*color_def*, *font=None*, *size=None*, *fill='black'*, *outline=None*, *outline_width=1*, *bg='white'*, *extend=False*, *unit=''*, *margins=None*, *minortick=0.0*, *nan_color=(0, 0, 0)*, *nan_check_color=(1, 1, 1)*, *nan_check_size=0*)

add_text(*txt*, ***kwargs*)

pydecorate.decorator_agg module

Aggdraw-based image decoration class.

class pydecorate.decorator_agg.**DecoratorAGG**(*image*)

Bases: *DecoratorBase*

Aggdraw-based image decoration class.

add_logo(*logo_path*, ***kwargs*)

add_scale(*colormap*, ***kwargs*)

add_text(*txt*, ***kwargs*)

pydecorate.decorator_base module

Base class and utilities for decorating images.

class pydecorate.decorator_base.**DecoratorBase**(*image*)

Bases: `object`

Base class for drawing decorations on an Image.

align_bottom()

align_left()

align_right()

align_top()

home()

new_line()

Set position of next decoration under the previously added decorations.

rewind()

set_style(***kwargs*)

write_horizontally()

write_vertically()

pydecorate.version module

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

pydecorate, 13
pydecorate.decorator, 12
pydecorate.decorator_agg, 13
pydecorate.decorator_base, 13
pydecorate.fonts, 11
pydecorate.tests, 12
pydecorate.tests.test_decorator_agg, 11
pydecorate.tests.test_legacy, 12
pydecorate.version, 13

INDEX

A

add_logo() (*pydecorate.decorator.Decorator method*), 12
add_logo() (*pydecorate.decorator_agg.DecoratorAGG method*), 13
add_scale() (*pydecorate.decorator.Decorator method*), 12
add_scale() (*pydecorate.decorator_agg.DecoratorAGG method*), 13
add_text() (*pydecorate.decorator.Decorator method*), 12
add_text() (*pydecorate.decorator_agg.DecoratorAGG method*), 13
align_bottom() (*pydecorate.decorator_base.DecoratorBase method*), 13
align_left() (*pydecorate.decorator_base.DecoratorBase method*), 13
align_right() (*pydecorate.decorator_base.DecoratorBase method*), 13
align_top() (*pydecorate.decorator_base.DecoratorBase method*), 13
assert_colorbar_increasing_tick_order() (*in module pydecorate.tests.test_decorator_agg*), 11
assert_colorbar_orientation_alignment() (*in module pydecorate.tests.test_decorator_agg*), 11

D

Decorator (*class in pydecorate.decorator*), 12
DecoratorAGG (*class in pydecorate.decorator_agg*), 13
DecoratorBase (*class in pydecorate.decorator_base*), 13

H

home() (*pydecorate.decorator_base.DecoratorBase method*), 13

M

module
    pydecorate, 13
    pydecorate.decorator, 12
    pydecorate.decorator_agg, 13
    pydecorate.decorator_base, 13
    pydecorate.fonts, 11
    pydecorate.tests, 12
    pydecorate.tests.test_decorator_agg, 11
    pydecorate.tests.test_legacy, 12
    pydecorate.version, 13

N

new_line() (*pydecorate.decorator_base.DecoratorBase method*), 13

P

pydecorate
    module, 13
pydecorate.decorator
    module, 12
pydecorate.decorator_agg
    module, 13
pydecorate.decorator_base
    module, 13
pydecorate.fonts
    module, 11
pydecorate.tests
    module, 12
pydecorate.tests.test_decorator_agg
    module, 11
pydecorate.tests.test_legacy
    module, 12
pydecorate.version
    module, 13

R

rewind() (*pydecorate.decorator_base.DecoratorBase method*), 13

S

set_style() (*pydeco-*

rate.decorator_base.DecoratorBase method), 13

T

test_add_logo() (*in module pydecorate.tests.test_decorator_agg*), 12
test_colorbar() (*in module pydecorate.tests.test_decorator_agg*), 12
test_style_retention() (*in module pydecorate.tests.test_legacy*), 12

W

write_horizontally() (*pydecorate.decorator_base.DecoratorBase method*), 13
write_vertically() (*pydecorate.decorator_base.DecoratorBase method*), 13