Colorzero Documentation

Release 1.0

Dave Jones

Mar 10, 2018

Contents

CHAPTER 1

Installation

1.1 Raspbian installation

On Raspbian[1], it is best to obtain colorzero via the `apt` utility:

```
$ sudo apt update
$ sudo apt install python-colorzero python3-colorzero
```

The usual apt upgrade method can be used to keep your installation up to date:

```
$ sudo apt update
$ sudo apt upgrade
```

To remove your installation:

```
$ sudo apt remove python-colorzero python3-colorzero
```

1.2 Ubuntu installation

If you are using Ubuntu[2], it is probably easiest to obtain colorzero from the author's PPA:

```
$ sudo add-apt-repository ppa://waveform/ppa
$ sudo apt update
$ sudo apt install python-colorzero python3-colorzero
```

The usual apt upgrade method can be used to keep your installation up to date:

```
$ sudo apt update
$ sudo apt upgrade
```

To remove your installation:

```
$ sudo apt remove python-colorzero python3-colorzero
```

[1] https://www.raspberrypi.org/downloads/raspbian/
[2] https://ubuntu.com/

1.3 Other platforms

On other platforms, it is probably easiest to obtain colorzero via the `pip` utility:

```
$ sudo pip install colorzero
$ sudo pip3 install colorzero
```

To upgrade your installation:

```
$ sudo pip install -U colorzero
$ sudo pip3 install -U colorzero
```

To remove your installation:

```
$ sudo pip remove colorzero
$ sudo pip3 remove colorzero
```

CHAPTER 2

Getting started

The `Color` (page 9) class is the main interface provided by colorzero. It can be constructed in a large variety of ways including with red, green, and blue components, "well known" color names (taken from CSS 3's extended color keywords[3]), HTML color specifications, and more. A selection of valid constructors is shown below:

```
>>> from colorzero import *
>>> Color('red')
<Color html="#ff0000" rgb=(1.0, 0.0, 0.0)>
>>> Color(1.0, 0.0, 0.0)
<Color html="#ff0000" rgb=(1.0, 0.0, 0.0)>
>>> Color(255, 0, 0)
<Color html="#ff0000" rgb=(1.0, 0.0, 0.0)>
>>> Color('#ff0000')
<Color html="#ff0000" rgb=(1.0, 0.0, 0.0)>
>>> Color('#f00')
<Color html="#ff0000" rgb=(1.0, 0.0, 0.0)>
```

Internally, colorzero always represents colors as red, green, and blue values between 0.0 and 1.0. `Color` (page 9) objects are tuple descendents. Crucially, this means they are *immutable*. Attempting to change the red, green, or blue attributes will fail:

```
>>> c = Color('red')
>>> c.red
Red(1.0)
>>> c.red = 0.5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
```

In order to manipulate a color, colorzero provides a simple series of classes which represent attributes of a color: `Red` (page 14), `Green` (page 15), `Blue` (page 15), `Hue` (page 15), `Lightness` (page 16), `Saturation` (page 15) and so on. You can use these classes in combination with Python's usual mathematical operators (addition, subtraction, multiplication, etc.) to manipulate a color. For example, continuing the example from above:

```
>>> c + Green(0.1)
<Color html='#ff1a00' rgb=(1, 0.1, 0)>
>>> c = c + Green(0.5)
```

[3] https://www.w3.org/TR/css3-color/#svg-color

```
>>> c
<Color html='#ff8000' rgb=(1, 0.5, 0)>
>>> c.lightness
Lightness(0.5)
>>> c = c * Lightness(0.5)
>>> c
<Color html='#804000' rgb=(0.5, 0.25, 0)>
```

Numerous attributes are provided to enable conversion of the RGB representation to other systems:

```
>>> c.rgb
RGB(r=0.5, g=0.25, b=0.0)
>>> c.rgb_bytes
RGB(r=128, g=64, b=0)
>>> c.rgb565
31200
>>> c.hls
HLS(h=0.08333333333333333, l=0.25, s=1.0)
>>> c.xyz
XYZ(x=0.10647471144683732, y=0.0819048964489466, z=0.010202272707313633)
>>> c.lab
Lab(l=34.376494620040376, a=23.890819210881016, b=44.69197916172735)
```

Equivalent constructors exist for all these systems:

```
>>> Color.from_rgb(0.5, 0.25, 0.0)
<Color html='#804000' rgb=(0.5, 0.25, 0)>
>>> Color.from_rgb_bytes(128, 64, 0)
<Color html='#804000' rgb=(0.501961, 0.25098, 0)>
>>> Color.from_rgb565(31200)
<Color html='#7b3d00' rgb=(0.483871, 0.238095, 0)>
>>> Color.from_hls(*c.hls)
<Color html='#804000' rgb=(0.5, 0.25, 0)>
>>> Color.from_xyz(*c.xyz)
<Color html='#7f4000' rgb=(0.5, 0.25, 0)>
>>> Color.from_lab(*c.lab)
<Color html='#7f4000' rgb=(0.5, 0.25, 0)>
```

Note that some conversions lose a certain amount of precision.

Methods are also provided to compare colors for similarity. The simplest algorithm (and the default) is "euclid" which calculates the difference as the distance between them by treating the r, g, b components as coordinates in a 3-dimensional space. The same color will have a distance of 0.0, whilst the largest possible difference is sqrt(3) (~1.732):

```
>>> c1 = Color('red')
>>> c2 = Color('green')
>>> c3 = c1 * Lightness(0.9)
>>> c1.difference(c2, 'euclid')
1.1189122525867927
>>> c1.difference(c2)
1.1189122525867927
>>> c1.difference(c3)
0.09999999999999998
```

Various Delta-E[4] algorithms (CIE1976, CIE1994, and CIEDE2000) are also provided. In these systems, 2.3 is considered a "just noticeable difference":

```
>>> c1.difference(c2, 'cie1976')
133.10729836196307
>>> c1.difference(c3, 'cie1976')
```

[4] https://en.wikipedia.org/wiki/Color_difference

```
9.60280542204272
>>> c1.difference(c2, 'cie1994g')
50.97596644678241
>>> c1.difference(c3, 'cie1994g')
5.484832836355026
>>> c1.difference(c2, 'ciede2000')
72.18229138962074
>>> c1.difference(c3, 'ciede2000')
5.490813507834904
```

CHAPTER 3

Development

The main GitHub repository for the project can be found at:

https://github.com/waveform80/colorzero

Anyone is more than welcome to open tickets to discuss bugs, new features, or just to ask usage questions (I find this useful for gauging what questions ought to feature in the FAQ, for example).

Even if you don't feel up to hacking on the code, I'd love to hear suggestions from people of what you'd like the API to look like (even if the code itself isn't particularly pythonic, the interface should be)!

3.1 Development installation

If you wish to develop colorzero itself, it is easiest to obtain the source by cloning the GitHub repository and then use the "develop" target of the Makefile which will install the package as a link to the cloned repository allowing in-place development (it also builds a tags file for use with vim/emacs with Exuberant's ctags utility). The following example demonstrates this method within a virtual Python environment:

```
$ sudo apt install lsb-release build-essential git git-core \
    exuberant-ctags virtualenvwrapper python-virtualenv python3-virtualenv
$ cd
$ mkvirtualenv -p /usr/bin/python3 colorzero
$ workon colorzero
(colorzero) $ git clone https://github.com/waveform80/colorzero.git
(colorzero) $ cd colorzero
(colorzero) $ make develop
```

To pull the latest changes from git into your clone and update your installation:

```
$ workon colorzero
(colorzero) $ cd ~/colorzero
(colorzero) $ git pull
(colorzero) $ make develop
```

To remove your installation, destroy the sandbox and the clone:

```
(colorzero) $ deactivate
$ rmvirtualenv colorzero
$ rm -fr ~/colorzero
```

3.2 Building the docs

If you wish to build the docs, you'll need a few more dependencies. Inkscape is used for conversion of SVGs to other formats, Graphviz is used for rendering certain charts, and TeX Live is required for building PDF output. The following command should install all required dependencies:

```
$ sudo apt install texlive-latex-recommended texlive-latex-extra \
    texlive-fonts-recommended graphviz inkscape
```

Once these are installed, you can use the "doc" target to build the documentation:

```
$ workon colorzero
(colorzero) $ cd ~/colorzero
(colorzero) $ make doc
```

The HTML output is written to `build/html` while the PDF output goes to `build/latex`.

3.3 Test suite

If you wish to run the colorzero test suite, follow the instructions in *Development installation* (page 7) above and then make the "test" target within the sandbox:

```
$ workon colorzero
(colorzero) $ cd ~/colorzero
(colorzero) $ make test
```

CHAPTER 4

API

The colorzero library includes a comprehensive `Color` (page 9) class which is capable of converting between numerous color representations and calculating color differences. Various ancillary classes can be used to manipulate aspects of a color.

4.1 Color

This the primary class in the package, and often the only class you'll need or want to interact with. It has an extremely flexible constructor, along with numerous explicit constructors, and attributes for conversion to other color systems.

class colorzero.**Color**

The Color class is a tuple which represents a color as linear red, green, and blue components.

The class has a flexible constructor which allows you to create an instance from any registered color system (see `color_conversion()`). There are also explicit constructors for every registered system that can convert (directly or indirectly) to linear RGB. For example, an instance of `Color` (page 9) can be constructed in any of the following ways:

```
>>> Color('#f00')
<Color html='#ff0000' rgb=(1, 0, 0)>
>>> Color('green')
<Color html='#008000' rgb=(0.0, 0.501961, 0.0)>
>>> Color(0, 0, 1)
<Color html='#0000ff' rgb=(0, 0, 1)>
>>> Color(h=0, s=1, v=0.5)
<Color html='#800000' rgb=(0.5, 0, 0)>
>>> Color(y=0.4, u=-0.05, v=0.615)
<Color html='#ff104c' rgb=(1, 0.0626644, 0.298394)>
```

The specific forms that the default constructor will accept are enumerated below:

Style	Description
Single scalar parameter	Equivalent to calling `Color.from_string()` (page 12), or `Color.from_rgb24()` (page 12).
Three positional parameters or a 3-tuple with no field names	Equivalent to calling `Color.from_rgb()` (page 12) if all three parameters are between 0.0 and 1.0, or `Color.from_rgb_bytes()` (page 12) otherwise.
Three named parameters, or a 3-tuple with fields "r", "g", "b"	
Three named parameters, or a 3-tuple with fields "red", "green", "blue"	
Three named parameters, or a 3-tuple with fields "y", "u", "v"	Equivalent to calling `Color.from_yuv()` (page 13) if "y" is between 0.0 and 1.0, "u" is between -0.436 and 0.436, and "v" is between -0.615 and 0.615, or `Color.from_yuv_bytes()` (page 13) otherwise.
Three named parameters, or a 3-tuple with fields "y", "i", "q"	Equivalent to calling `Color.from_yiq()` (page 13).
Three named parameters, or a 3-tuple with fields "h", "l", "s"	Equivalent to calling `Color.from_hls()` (page 12).
Three named parameters, or a 3-tuple with fields "hue", "lightness", "saturation"	
Three named parameters, or a 3-tuple with fields "h", "s", "v"	Equivalent to calling `Color.from_hsv()` (page 12)
Three named parameters, or a 3-tuple with fields "hue", "saturation", "value"	
Three named parameters, or a 3-tuple with fields "x", "y", "z"	Equivalent to calling `Color.from_cie_xyz()`
Three named parameters, or a 3-tuple with fields "l", "a", "b"	Equivalent to calling `Color.from_cie_lab()`
Three named parameters, or a 3-tuple with fields "l", "u", "v"	Equivalent to calling `Color.from_cie_luv()`

If the constructor parameters do not conform to any of the variants in the table above, a `ValueError`[5] will be raised.

Internally, the color is *always* represented as 3 `float`[6] values corresponding to the red, green, and blue components of the color. These values take a value from 0.0 to 1.0 (least to full intensity). The class provides several attributes which can be used to convert one color system into another:

```
>>> Color('#f00').hls
HLS(h=0.0, l=0.5, s=1.0)
>>> Color.from_string('green').hue
Hue(deg=120.0)
>>> Color.from_rgb_bytes(0, 0, 255).yuv
YUV(y=0.114, u=0.436, v=-0.10001426533523537)
```

[5] https://docs.python.org/3.5/library/exceptions.html#ValueError
[6] https://docs.python.org/3.5/library/functions.html#float

As `Color` (page 9) derives from tuple, instances are immutable. While this provides the advantage that they can be used in a `set`[7] or as keys of a `dict`[8], it does mean that colors themselves cannot be *directly* manipulated (e.g. by setting the red component).

However, several auxilliary classes in the module provide the ability to perform simple transformations of colors via operators which produce a new `Color` (page 9) instance. For example, you can add, subtract, and multiply colors directly:

```
>>> Color('red') + Color('blue')
<Color html='#ff00ff' rgb=(1, 0, 1)>
>>> Color('magenta') - Color('red')
<Color html='#0000ff' rgb=(0, 0, 1)>
```

Values are clipped to ensure the resulting color is still valid:

```
>>> Color('#ff00ff') + Color('#ff0000')
<Color html='#ff00ff' rgb=(1, 0, 1)>
```

You can wrap numbers in constructors like `Red` (page 14) (or obtain elements of existing colors), then add, subtract, or multiply them with a `Color` (page 9):

```
>>> Color('red') - Red(0.5)
<Color html='#800000' rgb=(0.5, 0, 0)>
>>> Color('green') + Color('grey').red
<Color html='#808000' rgb=(0.501961, 0.501961, 0)>
```

You can even manipulate non-primary attributes like hue, saturation, and lightness with standard addition, subtraction or multiplication operators:

```
>>> Color.from_hls(0.5, 0.5, 1.0)
<Color html='#00ffff' rgb=(0, 1, 1)>
>>> Color.from_hls(0.5, 0.5, 1.0) * Lightness(0.8)
<Color html='#00cccc' rgb=(0, 0.8, 0.8)>
>>> (Color.from_hls(0.5, 0.5, 1.0) * Lightness(0.8)).hls
HLS(h=0.5, l=0.4, s=1.0)
```

In the last example above, a `Color` (page 9) instance is constructed from HLS (hue, lightness, saturation) values with a lightness of 0.5. This is multiplied by a `Lightness` (page 16) a value of 0.8 which constructs a new `Color` (page 9) with the same hue and saturation, but a lightness of 0.4 (0.8 * 0.5).

If an instance is converted to a string (with `str()`) it will return a string containing the 7-character HTML code for the color (e.g. "#ff0000" for red). As can be seen in the examples above, a similar representation is included for the output of `repr()`[9].

difference(*other*, *method='euclid'*)

Determines the difference between this color and *other* using the specified *method*. The *method* is specified as a string, and the following methods are valid:

- 'euclid' - This is the default method. Calculate the Euclidian distance[10]. This is by far the fastest method, but also the least accurate in terms of human perception.
- 'cie1976' - Use the CIE 1976[11] formula for calculating the difference between two colors in CIE Lab space.
- 'cie1994g' - Use the CIE 1994[12] formula with the "graphic arts" bias for calculating the difference.
- 'cie1994t' - Use the CIE 1994[13] forumula with the "textiles" bias for calculating the difference.

[7] https://docs.python.org/3.5/library/stdtypes.html#set
[8] https://docs.python.org/3.5/library/stdtypes.html#dict
[9] https://docs.python.org/3.5/library/functions.html#repr
[10] https://en.wikipedia.org/wiki/Euclidean_distance
[11] https://en.wikipedia.org/wiki/Color_difference#CIE76
[12] https://en.wikipedia.org/wiki/Color_difference#CIE94
[13] https://en.wikipedia.org/wiki/Color_difference#CIE94

- 'cie2000' - Use the CIEDE 2000[14] formula for calculating the difference.

Note that the Euclidian distance will be significantly different to the other calculations; effectively this just measures the distance between the two colors by treating them as coordinates in a three dimensional Euclidian space. All other methods are means of calculating a Delta E[15] value in which 2.3 is considered a just-noticeable difference[16] (JND).

classmethod from_cmy (*c*, *m*, *y*)
Construct a `Color` (page 9) from CMY[17] (cyan, magenta, yellow) floats between 0.0 and 1.0.

Note: This conversion uses the basic subtractive method which is not accurate for color reproduction on print devices. See the Color FAQ[18] for more information.

classmethod from_cmyk (*c*, *m*, *y*, *k*)
Construct a `Color` (page 9) from CMYK[19] (cyan, magenta, yellow, black) floats between 0.0 and 1.0.

Note: This conversion uses the basic subtractive method which is not accurate for color reproduction on print devices. See the Color FAQ[20] for more information.

classmethod from_hls (*h*, *l*, *s*)
Construct a `Color` (page 9) from HLS[21] (hue, lightness, saturation) floats between 0.0 and 1.0.

classmethod from_hsv (*h*, *s*, *v*)
Construct a `Color` (page 9) from HSV[22] (hue, saturation, value) floats between 0.0 and 1.0.

classmethod from_lab (*l*, *a*, *b*)
Construct a `Color` (page 9) from (L*, a*, b*) float values representing a color in the CIE Lab color space[23]. The conversion assumes the sRGB working space with reference white D65.

classmethod from_luv (*l*, *u*, *v*)
Construct a `Color` (page 9) from (L*, u*, v*) float values representing a color in the CIE Luv color space[24]. The conversion assumes the sRGB working space with reference white D65.

classmethod from_rgb (*r*, *g*, *b*)
Construct a `Color` (page 9) from three linear RGB[25] float values between 0.0 and 1.0.

classmethod from_rgb24 (*n*)
Construct a `Color` (page 9) from an unsigned 24-bit integer number of the form 0x00BBGGRR.

classmethod from_rgb565 (*n*)
Construct a `Color` (page 9) from an unsigned 16-bit integer number in RGB565 format.

classmethod from_rgb_bytes (*r*, *g*, *b*)
Construct a `Color` (page 9) from three RGB[26] byte values between 0 and 255.

classmethod from_string (*s*)
Construct a `Color` (page 9) from a 4 or 7 character CSS-like representation (e.g. "#f00" or "#ff0000" for red), or from one of the named colors (e.g. "green" or "wheat") from the CSS standard[27]. Any

[14] https://en.wikipedia.org/wiki/Color_difference#CIEDE2000
[15] https://en.wikipedia.org/wiki/Color_difference
[16] https://en.wikipedia.org/wiki/Just-noticeable_difference
[17] https://en.wikipedia.org/wiki/CMYK_color_model
[18] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[19] https://en.wikipedia.org/wiki/CMYK_color_model
[20] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[21] https://en.wikipedia.org/wiki/HSL_and_HSV
[22] https://en.wikipedia.org/wiki/HSL_and_HSV
[23] https://en.wikipedia.org/wiki/Lab_color_space
[24] https://en.wikipedia.org/wiki/CIELUV
[25] https://en.wikipedia.org/wiki/RGB_color_space
[26] https://en.wikipedia.org/wiki/RGB_color_space
[27] http://www.w3.org/TR/css3-color/#svg-color

other string format will result in a `ValueError`[28].

classmethod from_xyz (*x*, *y*, *z*)

Construct a *Color* (page 9) from (X, Y, Z) float values representing a color in the CIE 1931 color space[29]. The conversion assumes the sRGB working space with reference white D65.

classmethod from_yiq (*y*, *i*, *q*)

Construct a *Color* (page 9) from three Y'IQ[30] float values. Y' can be between 0.0 and 1.0, while I and Q can be between -1.0 and 1.0.

classmethod from_yuv (*y*, *u*, *v*)

Construct a *Color* (page 9) from three Y'UV[31] float values. The Y value may be between 0.0 and 1.0. U may be between -0.436 and 0.436, while V may be between -0.615 and 0.615.

classmethod from_yuv_bytes (*y*, *u*, *v*)

Construct a *Color* (page 9) from three Y'UV[32] byte values between 0 and 255. The U and V values are biased by 128 to prevent negative values as is typical in video applications. The Y value is biased by 16 for the same purpose.

cmy

Returns a 3-tuple of (cyan, magenta, yellow) float values (between 0.0 and 1.0).

Note: This conversion uses the basic subtractive method which is not accurate for color reproduction on print devices. See the Color FAQ[33] for more information.

cmyk

Returns a 4-tuple of (cyan, magenta, yellow, black) float values (between 0.0 and 1.0).

Note: This conversion uses the basic subtractive method which is not accurate for color reproduction on print devices. See the Color FAQ[34] for more information.

hls

Returns a 3-tuple of (hue, lightness, saturation) float values (between 0.0 and 1.0).

hsv

Returns a 3-tuple of (hue, saturation, value) float values (between 0.0 and 1.0).

html

Returns the color as a string in HTML #RRGGBB format.

hue

Returns the hue of the color as a *Hue* (page 15) instance which can be used in operations with other *Color* (page 9) instances.

lab

Returns a 3-tuple of (L*, a*, b*) float values representing the color in the CIE Lab color space[35] with the D65 standard illuminant[36].

lightness

Returns the lightness of the color as a *Lightness* (page 16) instance which can be used in operations with other *Color* (page 9) instances.

[28] https://docs.python.org/3.5/library/exceptions.html#ValueError
[29] https://en.wikipedia.org/wiki/CIE_1931_color_space
[30] https://en.wikipedia.org/wiki/YIQ
[31] https://en.wikipedia.org/wiki/YUV
[32] https://en.wikipedia.org/wiki/YUV
[33] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[34] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[35] https://en.wikipedia.org/wiki/Lab_color_space
[36] https://en.wikipedia.org/wiki/Illuminant_D65

luma
: Returns the luma[37] of the color as a *Luma* (page 16) instance which can be used in operations with other *Color* (page 9) instances.

luv
: Returns a 3-tuple of (L*, u*, v*) float values representing the color in the CIE Luv color space[38] with the D65 standard illuminant[39].

rgb
: Return a simple 3-tuple of (r, g, b) float values in the range 0.0 <= n <= 1.0.

Note: The *Color* (page 9) class can already be treated as such a 3-tuple but for the cases where you want a straight `namedtuple()`[40] this property is available.

rgb565
: Returns an unsigned 16-bit integer number representing the color in the RGB565 encoding.

rgb_bytes
: Returns a 3-tuple of (red, green, blue) byte values.

saturation
: Returns the saturation of the color as a *Saturation* (page 15) instance which can be used in operations with other *Color* (page 9) instances.

xyz
: Returns a 3-tuple of (X, Y, Z) float values representing the color in the CIE 1931 color space[41]. The conversion assumes the sRGB working space, with reference white D65.

yiq
: Returns a 3-tuple of (y, i, q) float values; y values can be between 0.0 and 1.0, whilst i and q values can be between -1.0 and 1.0.

yuv
: Returns a 3-tuple of (y, u, v) float values; Y values can be between 0.0 and 1.0, U values are between -0.436 and 0.436, and V values are between -0.615 and 0.615.

yuv_bytes
: Returns a 3-tuple of (y, u, v) byte values. Y values are biased by 16 in the result to prevent negatives. U and V values are biased by 128 for the same purpose.

4.2 Manipulation Classes

These manipulation classes are used in conjunction with the standard arithmetic addition, subtraction, and multiplication operators to calculate new *Color* (page 9) instances.

class colorzero.**Red**
: Represents the red component of a *Color* (page 9) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the `Color.red` attribute. Addition, subtraction, and multiplication are supported with *Color* (page 9) instances. For example:

```
>>> Color.from_rgb(0, 0, 0) + Red(0.5)
<Color html='#800000' rgb=(0.5, 0, 0)>
>>> Color('#f00') - Color('#900').red
<Color html='#660000' rgb=(0.4, 0, 0)>
>>> (Red(0.1) * Color('red')).red
Red(0.1)
```

[37] https://en.wikipedia.org/wiki/Luma_(video)
[38] https://en.wikipedia.org/wiki/CIELUV
[39] https://en.wikipedia.org/wiki/Illuminant_D65
[40] https://docs.python.org/3.5/library/collections.html#collections.namedtuple
[41] https://en.wikipedia.org/wiki/CIE_1931_color_space

class colorzero.Green

Represents the green component of a *Color* (page 9) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the `Color.green` attribute. Addition, subtraction, and multiplication are supported with *Color* (page 9) instances. For example:

```
>>> Color(0, 0, 0) + Green(0.1)
<Color html='#001a00' rgb=(0, 0.1, 0)>
>>> Color.from_yuv(1, -0.4, -0.6) - Green(1)
<Color html='#510030' rgb=(0.316098, 0, 0.187156)>
>>> (Green(0.5) * Color('white')).rgb
RGB(r=1.0, g=0.5, b=1.0)
```

class colorzero.Blue

Represents the blue component of a *Color* (page 9) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the `Color.blue` attribute. Addition, subtraction, and multiplication are supported with *Color* (page 9) instances. For example:

```
>>> Color(0, 0, 0) + Blue(0.2)
<Color html='#000033' rgb=(0, 0, 0.2)>
>>> Color.from_hls(0.5, 0.5, 1.0) - Blue(1)
<Color html='#00ff00' rgb=(0, 1, 0)>
>>> Blue(0.9) * Color('white')
<Color html='#ffffe6' rgb=(1, 1, 0.9)>
```

class colorzero.Hue

Represents the hue of a *Color* (page 9) for use in transformations. Instances of this class can be constructed directly with a float value in the range [0.0, 1.0) representing an angle around the HSL hue wheel[42]. As this is a circular mapping, 0.0 and 1.0 effectively mean the same thing, i.e. out of range values will be normalized into the range [0.0, 1.0).

The class can also be constructed with the keyword arguments `deg` or `rad` if you wish to specify the hue value in degrees or radians instead, respectively. Instances can also be constructed by querying the *Color.hue* (page 13) attribute.

Addition, subtraction, and multiplication are supported with *Color* (page 9) instances. For example:

```
>>> Color(1, 0, 0).hls
HLS(h=0.0, l=0.5, s=1.0)
>>> (Color(1, 0, 0) + Hue(deg=180)).hls
HLS(h=0.5, l=0.5, s=1.0)
```

Note that whilst multiplication by a *Hue* (page 15) doesn't make much sense, it is still supported. However, the circular nature of a hue value can lead to suprising effects. In particular, since 1.0 is equivalent to 0.0 the following may be observed:

```
>>> (Hue(1.0) * Color.from_hls(0.5, 0.5, 1.0)).hls
HLS(h=0.0, l=0.5, s=1.0)
```

deg

Returns the hue as a value in degrees with the range 0.0 <= n < 360.0.

rad

Returns the hue as a value in radians with the range 0.0 <= n < 2π.

class colorzero.Saturation

Represents the saturation of a *Color* (page 9) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the *Color.saturation* (page 14) attribute. Addition, subtraction, and multiplication are supported with *Color* (page 9) instances. For example:

[42] https://en.wikipedia.org/wiki/Hue

```
>>> Color(0.9, 0.9, 0.6) + Saturation(0.1)
<Color html='#ecec93' rgb=(0.925, 0.925, 0.575)>
>>> Color('red') - Saturation(1)
<Color html='#808080' rgb=(0.5, 0.5, 0.5)>
>>> Saturation(0.5) * Color('wheat')
<Color html='#e4d9c3' rgb=(0.896078, 0.85098, 0.766667)>
```

class colorzero.**Lightness**

Represents the lightness of a *Color* (page 9) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the *Color.lightness* (page 13) attribute. Addition, subtraction, and multiplication are supported with *Color* (page 9) instances. For example:

```
>>> Color(0, 0, 0) + Lightness(0.1)
<Color html='#1a1a1a' rgb=(0.1, 0.1, 0.1)>
>>> Color.from_rgb_bytes(0x80, 0x80, 0) - Lightness(0.2)
<Color html='#1a1a00' rgb=(0.101961, 0.101961, 0)>
>>> Lightness(0.9) * Color('wheat')
<Color html='#f0ce8e' rgb=(0.94145, 0.806785, 0.555021)>
```

class colorzero.**Luma**

Represents the luma of a *Color* (page 9) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the `Color.yuv.y` attribute. Addition, subtraction, and multiplication are supported with *Color* (page 9) instances. For example:

```
>>> Color(0, 0, 0) + Luma(0.1)
<Color html='#1a1a1a' rgb=(0.1, 0.1, 0.1)>
>>> Color('red') * Luma(0.5)
<Color html='#d90000' rgb=(0.8505, 0, 0)>
```

CHAPTER 5

Change log

5.1 Release 1.0 (2018-03-10)

1.0 is the first release after breaking the library out of the picamera[43] project. As this is a 1.x release, API stability will be maintained.

[43] https://github.com/waveform80/picamera

CHAPTER 6

License

[44] dave@waveform.org.uk

Index