colorzero 2.0 Documentation

Dave Jones

Mar 16, 2021

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 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 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 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:

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

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

1.3 Other platforms

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

```
$ sudo pip3 install colorzero
```

To upgrade your installation:

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

To remove your installation:

```
$ sudo pip3 remove colorzero
```

CHAPTER 2

Getting started

The *Color* (page 7) 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 7) 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 15), *Green* (page 15), *Blue* (page 16), *Hue* (page 16), *Lightness* (page 17), *Saturation* (page 16) 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:

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

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

The `repr()`[4] output of *Color* (page 7) is relatively verbose by default, but this can be customized via the *Color.repr_style* (page 9) class attribute:

```
>>> c = Color('red')
>>> c
<Color html="#ff0000" rgb=(1.0, 0.0, 0.0)>
>>> Color.repr_style = 'html'
>>> c
Color('#ff0000')
>>> Color.repr_style = 'rgb'
>>> c
Color(1, 0, 0)
```

If you have a terminal capable of color output (usually this means an actual terminal, not those integrated

[4] https://docs.python.org/3.5/library/functions.html#repr

into applications like IDLE, Thonny, etc.), you can also preview colors with this facility (the output below shows the ANSI codes produced, but the documentation system won't reproduce the colored output):

```
>>> Color.repr_style = 'term256'
>>> c
<Color ### rgb=(1, 0, 0)>
>>> repr(c)
'<Color \x1b[38;5;9m###\x1b[0m rgb=(1, 0, 0)>'
>>> Color.repr_style = 'term16m'
>>> c
<Color ### rgb=(1, 0, 0)>
>>> repr(c)
'<Color \x1b[38;2;255;0;0m###\x1b[0m rgb=(1, 0, 0)>'
```

These ANSI codes can also be generated by using colors with `str.format()`[5]. For example:

```
>>> '{c:16m}Red{c:0} Alert!'.format(c=Color('red'))
'\x1b[38;2;255;0;0mRed\x1b[0m Alert!'
```

See *Format Strings* (page 14) for more information.

A method (*gradient()* (page 12)) is provided to generate gradients which fade from one color to another. The result is a generator, which must be iterated over if you want all the results:

```
>>> Color.repr_style = 'term16m'
>>> for c in Color('red').gradient(Color('green')):
...     print(repr(c))
...
<Color ### rgb=(1, 0, 0)>
<Color ### rgb=(0.888889, 0.0557734, 0)>
<Color ### rgb=(0.777778, 0.111547, 0)>
<Color ### rgb=(0.666667, 0.16732, 0)>
<Color ### rgb=(0.555556, 0.223094, 0)>
<Color ### rgb=(0.444444, 0.278867, 0)>
<Color ### rgb=(0.333333, 0.334641, 0)>
<Color ### rgb=(0.222222, 0.390414, 0)>
<Color ### rgb=(0.111111, 0.446187, 0)>
<Color ### rgb=(0, 0.501961, 0)>
```

In a color-capable terminal, the "###" above will appear to fade between the two specified colors.

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

[5] https://docs.python.org/3.5/library/stdtypes.html#str.format
[6] https://en.wikipedia.org/wiki/Color_difference

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

These algorithms are also available as straight-forward functions:

```
>>> cie1976(c1, c2)
133.10729836196307
>>> ciede2000(c1, c3)
5.490813507834904
```

CHAPTER 3

API

The colorzero library includes a comprehensive *Color* (page 7) 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.

3.1 Color Class

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 built-in color system. There are also explicit constructors for every known system that can convert (directly or indirectly) to linear RGB. For example, an instance of *Color* (page 7) 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 11).
Three positional parameters or a 3-tuple with no field names	Equivalent to calling *Color.from_rgb()* (page 11) 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 12) 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 12) otherwise.
Three named parameters, or a 3-tuple with fields "y", "i", "q"	Equivalent to calling *Color.from_yiq()* (page 12).
Three named parameters, or a 3-tuple with fields "h", "l", "s"	Equivalent to calling *Color.from_hls()* (page 11).
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 11)
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_xyz()* (page 12)
Three named parameters, or a 3-tuple with fields "l", "a", "b"	Equivalent to calling *Color.from_lab()* (page 11)
Three named parameters, or a 3-tuple with fields "l", "u", "v"	Equivalent to calling *Color.from_luv()* (page 11)

If the constructor parameters do not conform to any of the variants in the table above, a `ValueError`[7] will be raised.

Internally, the color is *always* represented as 3 `float`[8] 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)
```

As *Color* (page 7) derives from tuple, instances are immutable. While this provides the advantage that they can be used in a `set`[9] or as keys of a `dict`[10], 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 7) instance. For example, you can add, subtract, and multiply colors directly:

[7] https://docs.python.org/3.5/library/exceptions.html#ValueError
[8] https://docs.python.org/3.5/library/functions.html#float
[9] https://docs.python.org/3.5/library/stdtypes.html#set
[10] https://docs.python.org/3.5/library/stdtypes.html#dict

```
>>> 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 15) (or obtain elements of existing colors), then add, subtract, or multiply them with a *Color* (page 7):

```
>>> 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 7) instance is constructed from HLS (hue, lightness, saturation) values with a lightness of 0.5. This is multiplied by a *Lightness* (page 17) a value of 0.8 which constructs a new *Color* (page 7) 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()`[11]. The output of `repr()`[12] can be customized by assigning values to *Color.repr_style* (page 9).

red

Return the red value as a *Red* (page 15) instance

green

Return the green value as a *Green* (page 15) instance

blue

Return the blue value as a *Blue* (page 16) instance

repr_style

Specifies the style of output returned when using `repr()`[13] against a *Color* (page 7) instance. This is an attribute of the class, not of instances. For example:

```
>>> Color('#f00')
<Color html='#ff0000' rgb=(1, 0, 0)>
>>> Color.repr_style = 'html'
>>> Color('#f00')
Color('#ff0000')
```

The following values are valid:

[11] https://docs.python.org/3.5/library/functions.html#repr
[12] https://docs.python.org/3.5/library/functions.html#repr
[13] https://docs.python.org/3.5/library/functions.html#repr

- 'default' - The style shown above
- 'term16m' - Similar to the default style, but instead of the HTML style being included, a swatch previewing the color is output. Note that the terminal must support 24-bit color ANSI codes[14] for this to work.
- 'term256' - Similar to 'termtrue', but uses the closest color that can be found in the standard 256-color xterm palette. Note that the terminal must support 8-bit color ANSI codes[15] for this to work.
- 'html' - Outputs a valid *Color* (page 7) constructor using the HTML style, e.g. `Color('#ff99bb')`
- 'rgb' - Outputs a valid *Color* (page 7) constructor using the floating point RGB values, e.g. `Color(1, 0.25, 0)`

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

Determines the difference between this color and *other* using the specified *method*.

Parameters

- **other** (Color (page 7)) – The color to compare this color to.
- **method** (*str*[16]) – The algorithm to use in the comparison. Valid values are:
  - 'euclid' - This is the default method. Calculate the Euclidian distance[17]. This is by far the fastest method, but also the least accurate in terms of human perception.
  - 'cie1976' - Use the CIE 1976[18] formula for calculating the difference between two colors in CIE Lab space.
  - 'cie1994g' - Use the CIE 1994[19] formula with the "graphic arts" bias for calculating the difference.
  - 'cie1994t' - Use the CIE 1994[20] forumula with the "textiles" bias for calculating the difference.
  - 'ciede2000' - Use the CIEDE 2000[21] formula for calculating the difference.

Returns A `float`[22] indicating how different the two colors are. 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[23] value in which 2.3 is considered a just-noticeable difference[24] (JND).

For example:

```
>>> Color('red').difference(Color('red'))
0.0
>>> Color('red').difference(Color('red'), method='cie1976')
0.0
>>> Color('red').difference(Color('#900'))
0.4
```

(continues on next page)

[14] https://en.wikipedia.org/wiki/ANSI_escape_code#24-bit
[15] https://en.wikipedia.org/wiki/ANSI_escape_code#8-bit
[16] https://docs.python.org/3.5/library/stdtypes.html#str
[17] https://en.wikipedia.org/wiki/Euclidean_distance
[18] https://en.wikipedia.org/wiki/Color_difference#CIE76
[19] https://en.wikipedia.org/wiki/Color_difference#CIE94
[20] https://en.wikipedia.org/wiki/Color_difference#CIE94
[21] https://en.wikipedia.org/wiki/Color_difference#CIEDE2000
[22] https://docs.python.org/3.5/library/functions.html#float
[23] https://en.wikipedia.org/wiki/Color_difference
[24] https://en.wikipedia.org/wiki/Just-noticeable_difference

(continued from previous page)

```
>>> Color('red').difference(Color('#900'), method='cie1976')
40.17063087142142
>>> Color('red').difference(Color('#900'), method='ciede2000')
21.078146289272155
>>> Color('red').difference(Color('blue'))
1.4142135623730951
>>> Color('red').difference(Color('blue'), method='cie1976')
176.31403908880046
```

Note: Instead of using this method, you may wish to simply use the various difference functions (*euclid()* (page 17), *cie1976()* (page 17), etc.) directly.

classmethod from_cmy(*c*, *m*, *y*)
Construct a *Color* (page 7) from CMY[25] (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[26] for more information.

classmethod from_cmyk(*c*, *m*, *y*, *k*)
Construct a *Color* (page 7) from CMYK[27] (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[28] for more information.

classmethod from_hls(*h*, *l*, *s*)
Construct a *Color* (page 7) from HLS[29] (hue, lightness, saturation) floats between 0.0 and 1.0.

classmethod from_hsv(*h*, *s*, *v*)
Construct a *Color* (page 7) from HSV[30] (hue, saturation, value) floats between 0.0 and 1.0.

classmethod from_lab(*l*, *a*, *b*)
Construct a *Color* (page 7) from (L*, a*, b*) float values representing a color in the CIE Lab color space[31]. The conversion assumes the sRGB working space with reference white D65.

classmethod from_luv(*l*, *u*, *v*)
Construct a *Color* (page 7) from (L*, u*, v*) float values representing a color in the CIE Luv color space[32]. The conversion assumes the sRGB working space with reference white D65.

classmethod from_rgb(*r*, *g*, *b*)
Construct a *Color* (page 7) from three linear RGB[33] float values between 0.0 and 1.0.

classmethod from_rgb24(*n*)
Construct a *Color* (page 7) from an unsigned 24-bit integer number of the form 0x00BBGGRR.

[25] https://en.wikipedia.org/wiki/CMYK_color_model
[26] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[27] https://en.wikipedia.org/wiki/CMYK_color_model
[28] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[29] https://en.wikipedia.org/wiki/HSL_and_HSV
[30] https://en.wikipedia.org/wiki/HSL_and_HSV
[31] https://en.wikipedia.org/wiki/Lab_color_space
[32] https://en.wikipedia.org/wiki/CIELUV
[33] https://en.wikipedia.org/wiki/RGB_color_space

classmethod from_rgb565(*n*)
Construct a *Color* (page 7) from an unsigned 16-bit integer number in RGB565 format.

classmethod from_rgb_bytes(*r*, *g*, *b*)
Construct a *Color* (page 7) from three RGB[34] byte values between 0 and 255.

classmethod from_string(*s*)
Construct a *Color* (page 7) 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[35]. Any other string format will result in a ValueError[36].

classmethod from_xyz(*x*, *y*, *z*)
Construct a *Color* (page 7) from (X, Y, Z) float values representing a color in the CIE 1931 color space[37]. The conversion assumes the sRGB working space with reference white D65.

classmethod from_yiq(*y*, *i*, *q*)
Construct a *Color* (page 7) from three Y'IQ[38] 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 7) from three Y'UV[39] 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 7) from three Y'UV[40] 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.

gradient(*other*, *steps=10*, *easing=<function linear>*)
Returns a generator which fades between this color and *other* in the specified number of *steps*.

Parameters

- **other** (Color (page 7)) – The color that will end the gradient (with the color the method is called upon starting the gradient)
- **steps** (*int*[41]) – The unqiue number of colors to include in the generated gradient. Defaults to 10 if unspecified.
- **easing** (*callable*) – A function which controls the speed of the progression. If specified, if must be a function which takes a single parameter, the number of *steps*, and yields a sequence of values between 0.0 (representing the start of the gradient) and 1.0 (representing the end). The default is *linear()* (page 18).

Returns A generator yielding *steps Color* (page 7) instances which fade from this color to *other*.

For example:

```
>>> Color.repr_style = 'html'
>>> print('\n'.join(
... repr(c) for c in
... Color('red').gradient(Color('green'))
... ))
Color('#ff0000')
Color('#e30e00')
Color('#c61c00')
```

(continues on next page)

[34] https://en.wikipedia.org/wiki/RGB_color_space
[35] http://www.w3.org/TR/css3-color/#svg-color
[36] https://docs.python.org/3.5/library/exceptions.html#ValueError
[37] https://en.wikipedia.org/wiki/CIE_1931_color_space
[38] https://en.wikipedia.org/wiki/YIQ
[39] https://en.wikipedia.org/wiki/YUV
[40] https://en.wikipedia.org/wiki/YUV
[41] https://docs.python.org/3.5/library/functions.html#int

(continued from previous page)

```
Color('#aa2b00')
Color('#8e3900')
Color('#714700')
Color('#555500')
Color('#396400')
Color('#1c7200')
Color('#008000')
```

New in version 1.1.

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[42] 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[43] 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 16) instance which can be used in operations with other *Color* (page 7) instances.

lab

Returns a 3-tuple of (L*, a*, b*) float values representing the color in the CIE Lab color space[44] with the D65 standard illuminant[45].

lightness

Returns the lightness of the color as a *Lightness* (page 17) instance which can be used in operations with other *Color* (page 7) instances.

luma

Returns the luma[46] of the color as a *Luma* (page 17) instance which can be used in operations with other *Color* (page 7) instances.

luv

Returns a 3-tuple of (L*, u*, v*) float values representing the color in the CIE Luv color space[47] with the D65 standard illuminant[48].

rgb

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

[42] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[43] http://poynton.ca/notes/colour_and_gamma/ColorFAQ.html#RTFToC24
[44] https://en.wikipedia.org/wiki/Lab_color_space
[45] https://en.wikipedia.org/wiki/Illuminant_D65
[46] https://en.wikipedia.org/wiki/Luma_(video)
[47] https://en.wikipedia.org/wiki/CIELUV
[48] https://en.wikipedia.org/wiki/Illuminant_D65

Note: The *Color* (page 7) class can already be treated as such a 3-tuple but for the cases where you want a straight `namedtuple()`[49] 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 16) instance which can be used in operations with other *Color* (page 7) instances.

xyz
Returns a 3-tuple of (X, Y, Z) float values representing the color in the CIE 1931 color space[50]. 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.

3.2 Format Strings

Instances of *Color* (page 7) can be used in format strings to output various representations of a color, including HTML sequences and ANSI escape sequences to color terminal output. Format specifications can be used to modify the output to support different terminal types. For example:

```
>>> red = Color('red')
>>> green = Color('green')
>>> blue = Color('#47b')
>>> print("{red:html}".format(red=red))
#ff0000
>>> print(repr("{red}Red{red:0} Alert!".format(red=red)))
'\\x1b[1;31mRed\\x1b[0m Alert!'
>>> print(repr("The grass is {green:16m}greener{green:0}.".format(
... green=green)))
'The grass is \\x1b[38;2;0;128;0mgreener\\x1b[0m.'
>>> print(repr("{blue:b16m}Blue skies{blue:0}".format(blue=blue)))
'\\x1b[48;2;68;119;187mBlue skies\\x1b[0m'
```

The format specification is one of:

- "html" - the color will be output as the common 7-character HTML represention of #RRGGBB where RR, GG, and BB are the red, green and blue components expressed as a single hexidecimal byte
- "css" or "cssrgb" - the color will be output in CSS' functional notation rgb(r, g, b) where r, g, and b are decimal representations of the red, green, and blue components in the range 0 to 255

[49] https://docs.python.org/3.5/library/collections.html#collections.namedtuple
[50] https://en.wikipedia.org/wiki/CIE_1931_color_space

- "csshsl" - the color will be output in CSS' function notation hue($h$deg, s%, l%) where h, s, and l are the hue (expressed in degrees), saturation, and lightness (expressed as percentages)
- One of the ANSI format specifications which consist of an optional foreground / background specifier (the letters "f" or "b") followed by an optional terminal type identifer, which is one of:
  - "8" - the default, indicating only the original 8 DOS colors are supported (technically, 16 foreground colors are supported via use of the "bold" style for "intense" colors)
  - "256" - indicates the terminal supports 256 colors via 8-bit color ANSI codes[51]
  - "16m" - indicating the terminal supports ~16 million colors via 24-bit color ANSI codes[52]

Alternately, "0" can be specified indicating that the style should be reset. If specified with the optional foreground / background specifier, "0" resets only the foreground / background color. If specified alone it resets all styles. More formally:

```
<term_fore_back> ::= "" | "f" | "b"
<term_type>      ::= "" | "0" | "8" | "256" | "16m"
<term>           ::= <term_fore_back> <term_type>
<html>           ::= "html"
<css>            ::= "css" ("rgb" | "hsl")?
<format_spec>    ::= <html> | <css> | <term>
```

New in version 1.1: The ability to output ANSI codes via format strings, and the customization of `repr()`[53] output.

New in version 1.2: The ability to output HTML and CSS representations via format strings

3.3 Manipulation Classes

These manipulation classes are used in conjunction with the standard arithmetic addition, subtraction, and multiplication operators to calculate new *Color* (page 7) instances.

class colorzero.Red

Represents the red component of a *Color* (page 7) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the *Color.red* (page 9) attribute. Addition, subtraction, and multiplication are supported with *Color* (page 7) 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)
```

class colorzero.Green

Represents the green component of a *Color* (page 7) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the *Color.green* (page 9) attribute. Addition, subtraction, and multiplication are supported with *Color* (page 7) 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)
```

(continues on next page)

[51] https://en.wikipedia.org/wiki/ANSI_escape_code#8-bit
[52] https://en.wikipedia.org/wiki/ANSI_escape_code#24-bit
[53] https://docs.python.org/3.5/library/functions.html#repr

(continued from previous page)

```
<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 7) for use in transformations. Instances of this class can be constructed directly with a float value, or by querying the *Color.blue* (page 9) attribute. Addition, subtraction, and multiplication are supported with *Color* (page 7) 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 7) 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[54]. 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 7) 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 16) 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 $0.0 <= n < 2\pi$.

class colorzero.Saturation

Represents the saturation of a *Color* (page 7) 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 7) instances. For example:

```
>>> Color(0.9, 0.9, 0.6) + Saturation(0.1)
<Color html='#ecec93' rgb=(0.925, 0.925, 0.575)>
>>> Color('red') - Saturation(1)
```

(continues on next page)

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

(continued from previous page)

```
<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 7) 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 7) 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 7) 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 7) 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)>
```

3.4 Difference Functions

`colorzero.euclid(`*color1*, *color2*`)`

Calculates color difference as a simple Euclidean distance[55] by treating the three components as spatial dimensions.

Note: This function will return considerably different values to the other difference functions. In particular, the maximum "difference" will be $\sqrt{3}$ which is much smaller than the output of the CIE functions.

`colorzero.cie1976(`*color1*, *color2*`)`

Calculates color difference according to the CIE 1976[56] formula. Effectively this is the Euclidean formula, but with CIE L*a*b* components instead of RGB.

`colorzero.cie1994g(`*color1*, *color2*`)`

Calculates color difference according to the CIE 1994[57] formula with the "textile" bias. See `cie1994()` for further information.

`colorzero.cie1994t(`*color1*, *color2*`)`

Calculates color difference according to the CIE 1994[58] formula with the "graphics" bias. See `cie1994()` for further information.

`colorzero.ciede2000(`*color1*, *color2*`)`

Calculates color difference according to the CIEDE 2000[59] formula. This is the most accurate

[55] https://en.wikipedia.org/wiki/Euclidean_distance
[56] https://en.wikipedia.org/wiki/Color_difference#CIE76
[57] https://en.wikipedia.org/wiki/Color_difference#CIE94
[58] https://en.wikipedia.org/wiki/Color_difference#CIE94
[59] https://en.wikipedia.org/wiki/Color_difference#CIEDE2000

algorithm currently implemented but also the most complex and slowest. Like CIE1994 it is largely based in CIE L*C*h* space, but with several modifications to account for perceptual uniformity flaws.

3.5 Easing Functions

These functions can be used with the *Color.gradient()* (page 12) method to control the progression of the fade between the two colors.

`colorzero.`**`linear`**(*steps*)
: Linear easing function; yields *steps* values between 0.0 and 1.0

`colorzero.`**`ease_in`**(*steps*)
: Quadratic ease-in function; yields *steps* values between 0.0 and 1.0

`colorzero.`**`ease_out`**(*steps*)
: Quadratic ease-out function; yields *steps* values between 0.0 and 1.0

`colorzero.`**`ease_in_out`**(*steps*)
: Quadratic ease-in-out function; yields *steps* values between 0.0 and 1.0

CHAPTER 4

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)!

4.1 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 build-essential git \
    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
```

4.2 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 texlive-xetex graphviz inkscape xindy
```

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`.

4.3 Test suite

If you wish to run the colorzero test suite, follow the instructions in *Development installation* (page 19) above and then make the "test" target within the sandbox:

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

CHAPTER 5

Change log

5.1 Release 2.0 (2021-03-15)

- Dropped Python 2.x support. Current Python support level is 3.5 and above.
- Added html and css format specifications to the *Color* (page 7) class' string-formatting capabilities.

5.2 Release 1.1 (2018-05-15)

- Added ability to generate ANSI codes with *Format Strings* (page 14).
- Added *Color.gradient()* (page 12) method.
- Exposed the various difference functions in the API (*euclid()* (page 17), *cie1976()* (page 17), etc).
- Various doc fixes and enhancements.

5.3 Release 1.0 (2018-03-10)

1.0 is the first release after breaking the library out of the picamera[60] project. As this is a 1.x release, API stability will be maintained.

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

CHAPTER 6

License

[61] dave@waveform.org.uk

Index

B

C

D

E

F

G

H

L

R

S

X

Y