Chroma Documentation

Release 0.2.0

Seena Burns

November 01, 2014

Chroma is a Python module for handling colors with ease.

Manipulating colors can quickly escalate into a tedious and complicated task, particularly when you become concerned with color systems beyond RGB. Chroma is here to provide a simple API to do the heavy lifting, so that you can stay focused on the important parts of your projects.

Before you ask, Chroma is BSD licensed, available on Github and PyPI.

CHAPTER 1

Features

- *Basic Color Tasks*
- Color Systems: *RGB*, *HEX*, *HLS*, *HSV*, *CMY and CMYK*
- *Alpha*
- *Blending (Additive and Subtractive Mixing)*

CHAPTER 2

Quickstart

Getting started with the power of Chroma is meant to be straightforward:

```
import chroma

# Create a color object
color = chroma.Color('#00FF00')

# Handling different color systems
color.cmy = (0.3, 0.7, 0.8)
color.rgb    # (0.7, 0.3, 0.2)
color.hls    # (0.0333, 0.45, 0.5556)

# Alpha
color.alpha = 0.5
color.hsv    # (0.03333, 0.7143, 0.7, 0.5)

# Color blending
color + chroma.Color("#FF00FF")
# #FF4DFF
```

And there you have it. The rest of this document describes Chroma's functionality and usage in detail.

2.1 Installation

Installation is as easy as:

```
pip install chroma
```

Or if you're an easy_install-er:

```
easy_install chroma
```

Chroma does not yet support Python 3, but, if you're interested, see *Contribute*

CHAPTER 3

Basic Color Tasks

At Chroma's core is the Color object. To create a color object, use the constructor, which accepts any of the available color systems:

`chroma.`**`Color`**`(`[*color_value = '#FFFFFF'*[, *format = 'HEX'*]]`)`

For example, to create a red Color object:

```
red = chroma.Color((1, 0, 0), 'RGB')
red
# #FF0000
```

Color objects can be compared with each other too.

```
red != chroma.Color('#00FF00')
# True
```

Manipulating Color objects can be easily achieved by using its properties. Every color system has a getter and setter for operating with that system.

```
red.hls = (10, 0.3, 0.9)
red.rgb256
# (145, 8, 8)
```

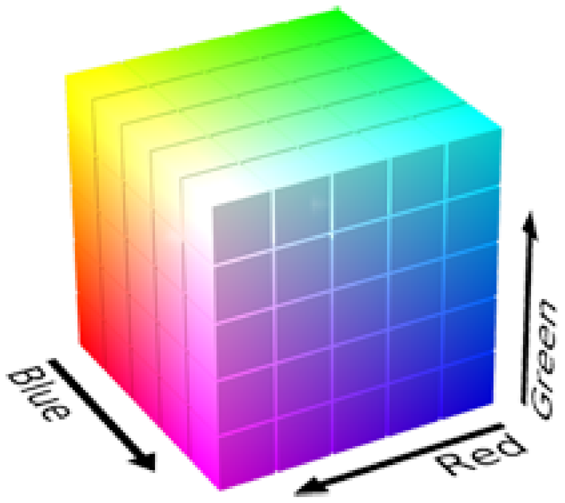
CHAPTER 4

Color Systems

Working with multiple color systems can be done using the getter and setter properties for each system.

Internally, colors are stored in rgb float format.

4.1 RGB - Red, Blue, Green



Chroma provides properties for RGB in both float and 256 tuple format. Color.rgb outputs float coordinates, ranging from 0 to 1, where 1 is white. Color.rgb256 outputs integer coordinates ranging from 0 to 255, where 255 is white.

RGB is used for *Additive mixing*.

If *Alpha* is active, alpha (float) will be appended to both rgb and rgb256. Likewise appending alpha to rgb and rgb256 setters will activate alpha on the color.

`chroma.Color.`**`rgb`**`()`

`chroma.Color.`**`rgb`**`(`*color_tuple*`)`

`chroma.Color.`**`rgb256`**`()`

`chroma.Color.`**`rgb256`**`(`*color_tuple*`)`

4.2 HEX - #rrggbb

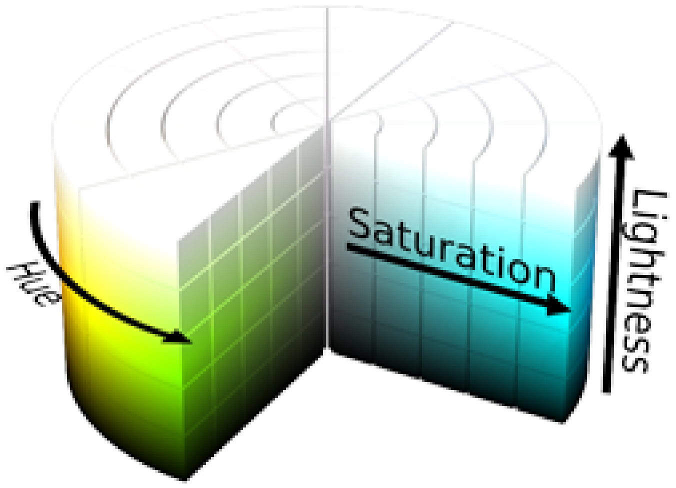
For convenience, a hexadecimal setter and getter are also provided, though hex representation is just a wrapper for RGB. The hexadecimal setter accepts a string in the format ('#RRGGBB'). Currently, Chroma does not support other hexadecimal representations.

As with the other RGB representations, alpha will be appended to the output if the alpha channel is active.

`chroma.Color.`**`hex`**`()`

`chroma.Color.`**`hex`**`(`*hex_string*`)`

4.3 HLS - Hue, Saturation, Lightness



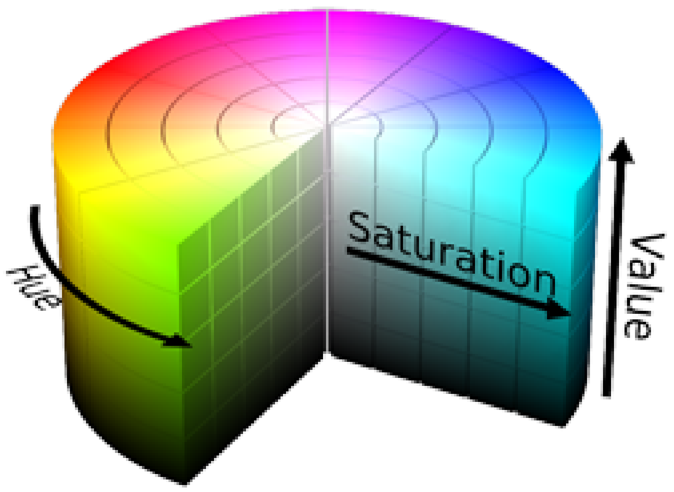
HLS (also known as HSL) is in the format (Hue °, Saturation %, Lightness %). Which means hue has a range of 0 - 360, while saturation and lightness have a range of 0 - 1.

As with RGB, Alpha will be appended if active.

`chroma.Color.`**`hls`**`()`

`chroma.Color.`**`hls`**`(`*color_tuple*`)`

4.4 HSV - Hue, Saturation, Value



Like HLS, HSV comes in the format (Hue °, Saturation %, Lightness %), so hue has a range of 0 - 360, but saturation and value have a range of 0 - 1.

NOTE: HSV saturation and HLS saturation are **NOT** the same:

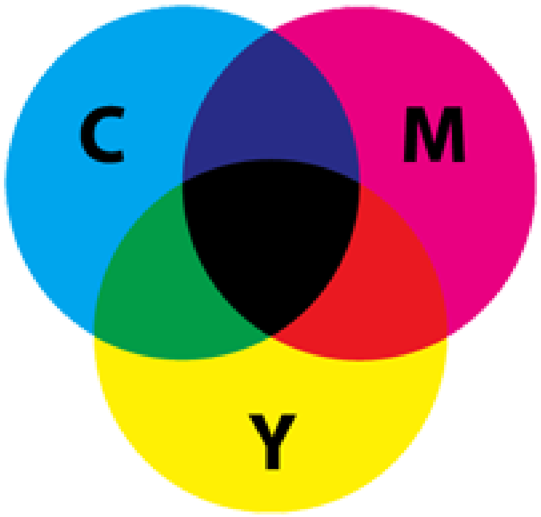
> "Under HSV model, adding white to a pure color reduces its saturation, while adding black to a pure color reduces it's value. Under the [HLS] model, adding white or black to a pure color simply moves you up and down the brightness axis, and only by adding combinations of white AND black can you alter the color's saturation."

As with RGB, Alpha will be appended if active.

`chroma.Color.`**`hsv`**`()`

`chroma.Color.`**`hsv`**`(`*color_tuple*`)`

4.5 CMY and CMYK - Cyan, Magenta, Yellow (and Black)



Chroma supports both CMY and CMYK input and output. CMY and CMYK are both in floating point form, ranging from 0 - 1, but are subtractive color models. This means, (1, 1, 1) represents black (not white) in CMY.

Unlike RGB, alpha will not be appended.

`chroma.Color.`**`cmy`**`()`

`chroma.Color.`**`cmy`**`(`*color_tuple*`)`

`chroma.Color.`**`cmyk`**`()`

`chroma.Color.`**`cmyk`**`(`*color_tuple*`)`

4.6 Alpha

Chroma supports an alpha channel, but it is unset by default. To activate alpha, you can pass in a color system with alpha appended (only RGB, RGB256, HEX, HLS, HSV) or set it using the property. Alpha ranges from 0 - 1 in floating point representation.

To turn off alpha, set it's value to None.

`chroma.Color.`**`alpha`**`()`

`chroma.Color.`**`alpha`**`(`*value*`)`

CHAPTER 5

Blending (Additive and Subtractive Mixing)

5.1 Additive (Light) Mixing

Additive mixing is a form of color blending that involves adding RGB values to create a lighter color mix from the two.

5.2 Subtractive (Dye) Mixing

Subtractive mixing can be thought of as taking a white light and overlaying color filters to ‘subtract’ colors from the light. The remaining light is the subtractive mix.

Specifically, it involves the difference of CMY values.

CHAPTER 6

Contribute

Chroma is under active development and could use your support. Even bug reports, feature suggestions and feedback can help push Chroma forward in the right direction.

Chroma is hosted on Github and there are a number of ideas of where to start in the issues section.

Index