

---

# **python-colormath Documentation**

*Release 2.0.0*

**Greg Taylor**

August 05, 2014



<b>1 Assorted Info</b>	<b>3</b>
<b>2 Topics</b>	<b>5</b>
2.1 Installation . . . . .	5
2.2 Color Objects . . . . .	5
2.3 Observers and Illuminants . . . . .	10
2.4 Color Conversions . . . . .	10
2.5 Delta E Equations . . . . .	11
2.6 ANSI and ISO Density . . . . .	12
2.7 Release Notes . . . . .	14
<b>3 Useful color math resources</b>	<b>17</b>



python-colormath is a simple Python module that spares the user from directly dealing with `color math`. Some features include:

- Support for a wide range of color spaces. A good chunk of the CIE spaces, RGB, HSL/HSV, CMY/CMYK, and many more.
- *Conversions* between the various color spaces. For example, XYZ to sRGB, Spectral to XYZ, CIE Lab to Adobe RGB.
- Calculation of *color difference*. All CIE Delta E functions, plus CMC.
- Chromatic adaptations (changing illuminants).
- RGB to hex and vice-versa.
- 16-bit RGB support.
- Runs on Python 2.7 and Python 3.3+.

**License:** python-colormath is licensed under the [BSD License](#).



## Assorted Info

---

- [Issue tracker](#) - Report bugs, ask questions, and share ideas here.
- [GitHub project](#) - Source code and issue tracking.
- [@gctaylor Twitter](#) - Tweets from the maintainer.
- [Greg Taylor's blog](#) - Occasional posts about color math and software development.





## 2.1 Installation

python-colormath currently requires Python 2.7 or Python 3.3+. There are no plans to add support for earlier versions of Python 2 or 3. The only other requirement is [NumPy](#).

For those on Linux/Unix Mac OS, the easiest route will be **pip** or **easy\_install**:

```
pip install colormath
```

If you are on Windows, you'll need to visit [NumPy](#), download their binary distribution, then install colormath.

## 2.2 Color Objects

python-colormath has support for many of the commonly used color spaces. These are represented by Color objects.

## 2.2.1 SpectralColor

```
class colormath.color_objects.SpectralColor (spec_340nm=0.0,          spec_350nm=0.0,
                                             spec_360nm=0.0,          spec_370nm=0.0,
                                             spec_380nm=0.0,          spec_390nm=0.0,
                                             spec_400nm=0.0,          spec_410nm=0.0,
                                             spec_420nm=0.0,          spec_430nm=0.0,
                                             spec_440nm=0.0,          spec_450nm=0.0,
                                             spec_460nm=0.0,          spec_470nm=0.0,
                                             spec_480nm=0.0,          spec_490nm=0.0,
                                             spec_500nm=0.0,          spec_510nm=0.0,
                                             spec_520nm=0.0,          spec_530nm=0.0,
                                             spec_540nm=0.0,          spec_550nm=0.0,
                                             spec_560nm=0.0,          spec_570nm=0.0,
                                             spec_580nm=0.0,          spec_590nm=0.0,
                                             spec_600nm=0.0,          spec_610nm=0.0,
                                             spec_620nm=0.0,          spec_630nm=0.0,
                                             spec_640nm=0.0,          spec_650nm=0.0,
                                             spec_660nm=0.0,          spec_670nm=0.0,
                                             spec_680nm=0.0,          spec_690nm=0.0,
                                             spec_700nm=0.0,          spec_710nm=0.0,
                                             spec_720nm=0.0,          spec_730nm=0.0,
                                             spec_740nm=0.0,          spec_750nm=0.0,
                                             spec_760nm=0.0,          spec_770nm=0.0,
                                             spec_780nm=0.0,          spec_790nm=0.0,
                                             spec_800nm=0.0,          spec_810nm=0.0,
                                             spec_820nm=0.0,          spec_830nm=0.0,
                                             observer='2', illuminant='d50')
```

A SpectralColor represents a spectral power distribution, as read by a spectrophotometer. Our current implementation has wavelength intervals of 10nm, starting at 340nm and ending at 830nm.

Spectral colors are the lowest level, most “raw” measurement of color. You may convert spectral colors to any other color space, but you can’t convert any other color space back to spectral.

See [Spectral power distribution](#) on Wikipedia for some higher level details on how these work.

### Parameters

- **observer** (*str*) – Observer angle. Either '2' or '10' degrees.
- **illuminant** (*str*) – See *Observers and Illuminants* for valid values.

## 2.2.2 LabColor

```
class colormath.color_objects.LabColor (lab_l, lab_a, lab_b, observer='2', illuminant='d50')
```

Represents a CIE Lab color. For more information on CIE Lab, see [Lab color space](#) on Wikipedia.

### Parameters

- **lab\_l** (*float*) – L coordinate.
- **lab\_a** (*float*) – a coordinate.
- **lab\_b** (*float*) – b coordinate.
- **observer** (*str*) – Observer angle. Either '2' or '10' degrees.
- **illuminant** (*str*) – See *Observers and Illuminants* for valid values.

### 2.2.3 LCHabColor

**class** `colormath.color_objects.LCHabColor` (*lch\_l, lch\_c, lch\_h, observer='2', illuminant='d50'*)

Represents an CIE LCH color that was converted to LCH by passing through CIE Lab. This differs from `LCHuvColor`, which was converted to LCH through CIE Luv.

See [Introduction to Colour Spaces](#) by Phil Cruse for an illustration of how CIE LCH differs from CIE Lab.

#### Parameters

- **lch\_l** (*float*) – L coordinate.
- **lch\_c** (*float*) – C coordinate.
- **lch\_h** (*float*) – H coordinate.
- **observer** (*str*) – Observer angle. Either '2' or '10' degrees.
- **illuminant** (*str*) – See [Observers and Illuminants](#) for valid values.

### 2.2.4 LCHuvColor

**class** `colormath.color_objects.LCHuvColor` (*lch\_l, lch\_c, lch\_h, observer='2', illuminant='d50'*)

Represents an CIE LCH color that was converted to LCH by passing through CIE Luv. This differs from `LCHabColor`, which was converted to LCH through CIE Lab.

See [Introduction to Colour Spaces](#) by Phil Cruse for an illustration of how CIE LCH differs from CIE Lab.

#### Parameters

- **lch\_l** (*float*) – L coordinate.
- **lch\_c** (*float*) – C coordinate.
- **lch\_h** (*float*) – H coordinate.
- **observer** (*str*) – Observer angle. Either '2' or '10' degrees.
- **illuminant** (*str*) – See [Observers and Illuminants](#) for valid values.

### 2.2.5 LuvColor

**class** `colormath.color_objects.LuvColor` (*luv\_l, luv\_u, luv\_v, observer='2', illuminant='d50'*)

Represents an Luv color.

#### Parameters

- **luv\_l** (*float*) – L coordinate.
- **luv\_u** (*float*) – u coordinate.
- **luv\_v** (*float*) – v coordinate.
- **observer** (*str*) – Observer angle. Either '2' or '10' degrees.
- **illuminant** (*str*) – See [Observers and Illuminants](#) for valid values.

## 2.2.6 XYZColor

**class** colormath.color\_objects.XYZColor(*xyz\_x, xyz\_y, xyz\_z, observer='2', illuminant='d50'*)  
Represents an XYZ color.

### Parameters

- **xyz\_x** (*float*) – X coordinate.
- **xyz\_y** (*float*) – Y coordinate.
- **xyz\_z** (*float*) – Z coordinate.
- **observer** (*str*) – Observer angle. Either '2' or '10' degrees.
- **illuminant** (*str*) – See *Observers and Illuminants* for valid values.

## 2.2.7 xyYColor

**class** colormath.color\_objects.xyYColor(*xyy\_x, xyy\_y, xyy\_Y, observer='2', illuminant='d50'*)  
Represents an xYy color.

### Parameters

- **xyy\_x** (*float*) – x coordinate.
- **xyy\_y** (*float*) – y coordinate.
- **xyy\_Y** (*float*) – Y coordinate.
- **observer** (*str*) – Observer angle. Either '2' or '10' degrees.
- **illuminant** (*str*) – See *Observers and Illuminants* for valid values.

## 2.2.8 sRGBColor

**class** colormath.color\_objects.sRGBColor(*rgb\_r, rgb\_g, rgb\_b, is\_upscaled=False*)  
Represents an sRGB color.

### Parameters

- **rgb\_r** (*float*) – R coordinate. 0...1. 1-255 if *is\_upscaled=True*.
- **rgb\_g** (*float*) – G coordinate. 0...1. 1-255 if *is\_upscaled=True*.
- **rgb\_b** (*float*) – B coordinate. 0...1. 1-255 if *is\_upscaled=True*.
- **is\_upscaled** (*bool*) – If False, RGB coordinate values are between 0.0 and 1.0. If True, RGB values are between 1 and 255.

## 2.2.9 AdobeRGBColor

**class** colormath.color\_objects.AdobeRGBColor(*rgb\_r, rgb\_g, rgb\_b, is\_upscaled=False*)  
Represents an Adobe RGB color.

### Parameters

- **rgb\_r** (*float*) – R coordinate. 0...1. 1-255 if *is\_upscaled=True*.
- **rgb\_g** (*float*) – G coordinate. 0...1. 1-255 if *is\_upscaled=True*.
- **rgb\_b** (*float*) – B coordinate. 0...1. 1-255 if *is\_upscaled=True*.

- **is\_upscaled** (*bool*) – If False, RGB coordinate values are between 0.0 and 1.0. If True, RGB values are between 1 and 255.

## 2.2.10 HSLColor

**class** colormath.color\_objects.**HSLColor** (*hsl\_h, hsl\_s, hsl\_l*)  
Represents an HSL color.

### Parameters

- **hsl\_h** (*float*) – H coordinate.
- **hsl\_s** (*float*) – S coordinate.
- **hsl\_l** (*float*) – L coordinate.

## 2.2.11 HSVColor

**class** colormath.color\_objects.**HSVColor** (*hsv\_h, hsv\_s, hsv\_v*)  
Represents an HSV color.

### Parameters

- **hsv\_h** (*float*) – H coordinate.
- **hsv\_s** (*float*) – S coordinate.
- **hsv\_v** (*float*) – V coordinate.

## 2.2.12 CMYColor

**class** colormath.color\_objects.**CMYColor** (*cmy\_c, cmy\_m, cmy\_y*)  
Represents a CMY color.

### Parameters

- **cmy\_c** (*float*) – C coordinate.
- **cmy\_m** (*float*) – M coordinate.
- **cmy\_y** (*float*) – Y coordinate.

## 2.2.13 CMYKColor

**class** colormath.color\_objects.**CMYKColor** (*cmyk\_c, cmyk\_m, cmyk\_y, cmyk\_k*)  
Represents a CMYK color.

### Parameters

- **cmyk\_c** (*float*) – C coordinate.
- **cmyk\_m** (*float*) – M coordinate.
- **cmyk\_y** (*float*) – Y coordinate.
- **cmyk\_k** (*float*) – K coordinate.

## 2.3 Observers and Illuminants

Illuminants and observer angles are used in all color spaces that use reflective (instead of transmissive) light. Here are a few brief overviews of what these are and what they do:

- Understanding Standard Illuminants in Color Measurement - Konica Minolta
- What is Meant by the Term “Observer Angle”? - XRite

To adjust the illuminants and/or observer angles on a color:

```
lab = LabColor(0.1, 0.2, 0.3, observer='10', illuminant='d65')
```

### 2.3.1 Two-degree observer angle

These illuminants can be used with `observer='2'`, for the color spaces that require illuminant/observer:

- 'a'
- 'b'
- 'c'
- 'd50'
- 'd55'
- 'd65'
- 'd75'
- 'e'
- 'f2'
- 'f7'
- 'f11'

### 2.3.2 Ten-degree observer angle

These illuminants can be used with `observer='10'`, for the color spaces that require illuminant/observer:

- 'd50'
- 'd55'
- 'd65'
- 'd75'

## 2.4 Color Conversions

Converting between color spaces is very simple with python-colormath. To see a full list of supported color spaces, see *Color Objects*.

All conversions happen through the `convert_color` function shown below. The original Color instance is passed in as the first argument, and the desired Color class (not an instance) is passed in as the second argument. If the conversion can be made, a new Color instance will be returned.

```
colormath.color_conversions.convert_color(color, target_cs, through_rgb_type=<class
                                         'colormath.color_objects.sRGBColor'>, *args,
                                         **kwargs)
```

Converts the color to the designated color space.

#### Parameters

- **color** – A Color instance to convert.
- **target\_cs** – The Color class to convert to. Note that this is not an instance, but a class.
- **through\_rgb\_type** (*BaseRGBColor*) – If during your conversion between your original and target color spaces you have to pass through RGB, this determines which kind of RGB to use. For example, XYZ->HSL. You probably don't need to specify this unless you have a special usage case.

**Returns** An instance of the type passed in as `target_cs`.

**Raises** `colormath.color_exceptions.UndefinedConversionError` if conversion between the two color spaces isn't possible.

## 2.4.1 Example

This is a simple example of a CIE Lab to CIE XYZ conversion. Refer to *Color Objects* for a full list of different color spaces that can be instantiated and converted between.

```
from colormath.color_objects import LabColor, XYZColor
from colormath.color_conversions import convert_color
```

```
lab = LabColor(0.903, 16.296, -2.22)
xyz = convert_color(lab, XYZColor)
```

Some color spaces require a trip through RGB during conversion. For example, to get from XYZ to HSL, we have to convert XYZ->RGB->HSL. The same could be said for XYZ to CMYK (XYZ->RGB->CMY->CMYK). Different RGB color spaces have different gamut sizes and capabilities, which can affect your converted color values.

sRGB is the default RGB color space due to its ubiquity. If you would like to use a different RGB space for a conversion, you can do something like this:

```
from colormath.color_objects import XYZColor, HSLColor, AdobeRGBColor
from colormath.color_conversions import convert_color
```

```
xyz = XYZColor(0.1, 0.2, 0.3)
hsl = convert_color(xyz, HSLColor, through_rgb_type=AdobeRGBColor)
# If you are going to convert back to XYZ, make sure you use the same
# RGB color space on the way back.
xyz2 = convert_color(hsl, XYZColor, through_rgb_type=AdobeRGBColor)
```

## 2.5 Delta E Equations

Delta E equations are used to put a number on the visual difference between two `LabColor` instances. While different lighting conditions, substrates, and physical condition can all introduce unexpected variables, these equations are a good rough starting point for comparing colors.

Each of the following Delta E functions has different characteristics. Some may be more suitable for certain applications than others. While it's outside the scope of this module's documentation to go into much detail, we link to relevant material when possible.

## 2.5.1 Example

```
from colormath.color_objects import LabColor
from colormath.color_diff import delta_e_cie1976

# Reference color.
color1 = LabColor(lab_l=0.9, lab_a=16.3, lab_b=-2.22)
# Color to be compared to the reference.
color2 = LabColor(lab_l=0.7, lab_a=14.2, lab_b=-1.80)
# This is your delta E value as a float.
delta_e = delta_e_cie1976(color1, color2)
```

## 2.5.2 Delta E CIE 1976

`colormath.color_diff.delta_e_cie1976` (*color1*, *color2*)  
Calculates the Delta E (CIE1976) of two colors.

## 2.5.3 Delta E CIE 1994

`colormath.color_diff.delta_e_cie1994` (*color1*, *color2*, *K\_L=1*, *K\_C=1*, *K\_H=1*, *K\_1=0.045*,  
*K\_2=0.015*)  
Calculates the Delta E (CIE1994) of two colors.

**K\_1:** 0.045 graphic arts 0.048 textiles

**K\_2:** 0.015 graphic arts 0.014 textiles

**K\_L:** 1 default 2 textiles

## 2.5.4 Delta E CIE 2000

`colormath.color_diff.delta_e_cie2000` (*color1*, *color2*, *Kl=1*, *Kc=1*, *Kh=1*)  
Calculates the Delta E (CIE2000) of two colors.

## 2.5.5 Delta E CMC

`colormath.color_diff.delta_e_cmc` (*color1*, *color2*, *pl=2*, *pc=1*)  
Calculates the Delta E (CMC) of two colors.

**CMC values** Acceptability: *pl=2*, *pc=1* Perceptability: *pl=1*, *pc=1*

## 2.6 ANSI and ISO Density

Density may be calculated from `LabColor` instances.

`colormath.density.auto_density` (*color*)

Given a `SpectralColor`, automatically choose the correct ANSI T filter. Returns a tuple with a string representation of the filter the calculated density.

**Parameters** *color* (*SpectralColor*) – The `SpectralColor` object to calculate density for.

**Return type** float



**Returns** The density value, with the filter selected automatically.

`colormath.density.ansi_density` (*color*, *density\_standard*)

Calculates density for the given `SpectralColor` using the spectral weighting function provided. For example, `ANSI_STATUS_T_RED`. These may be found in `colormath.density_standards`.

**Parameters**

- **color** (*SpectralColor*) – The `SpectralColor` object to calculate density for.
- **std\_array** (*numpy.ndarray*) – NumPy array of filter of choice from `colormath.density_standards`.

**Return type** float

**Returns** The density value for the given color and density standard.

## 2.6.1 Example

```
from colormath.color_objects import SpectralColor
from colormath.density import auto_density, ansi_density
from colormath.density_standards import ANSI_STATUS_T_RED

# Omitted the full spectral kwargs for brevity.
color = SpectralColor(spec_340nm=0.08, ...)
# ANSI T Density for the spectral color.
density = auto_density(color)

# Or maybe we want to specify which filter to use.
red_density = ansi_density(color, ANSI_STATUS_T_RED)
```

## 2.6.2 Valid Density Constants

The following density constants within `colormath.density_standards` can be passed to `colormath.density.ansi_density()`:

- `ANSI_STATUS_A_RED`
- `ANSI_STATUS_A_GREEN`
- `ANSI_STATUS_A_BLUE`
- `ANSI_STATUS_E_RED`
- `ANSI_STATUS_E_GREEN`
- `ANSI_STATUS_E_BLUE`
- `ANSI_STATUS_M_RED`
- `ANSI_STATUS_M_GREEN`
- `ANSI_STATUS_M_BLUE`
- `ANSI_STATUS_T_RED`
- `ANSI_STATUS_T_GREEN`
- `ANSI_STATUS_T_BLUE`
- `TYPE1`
- `TYPE2`

- ISO\_VISUAL

## 2.7 Release Notes

### 2.7.1 2.0.0

#### Backwards Incompatible changes

- Minimum Python version is now 2.7.
- `ColorBase.convert_to()` is no more. Use `colormath.color_conversions.convert_color()` instead. API isn't as spiffy looking, but it's a lot less magical now.
- Completely re-worked RGB handling. Each RGB space now has its own class, inheriting from `BaseRGBColor`. Consequently, `RGBColor` is no more. In most cases, you can substitute `RGBColor` with `sRGBColor` during your upgrade.
- RGB channels are now `[0..1]` instead of `[1..255]`. Can use `BaseRGBColor.get_upscaled_value_tuple()` to get the upscaled values.
- `BaseRGBColor.set_from_rgb_hex()` was replaced with a `BaseRGBColor.new_from_rgb_hex()`, which returns a properly formed `sRGBColor` object.
- `BaseRGBColor` no longer accepts `observer` or `illuminant` kwargs.
- `HSL` no longer accepts `observer`, `illuminant` or `rgb_type` kwargs.
- `HSV` no longer accepts `observer`, `illuminant` or `rgb_type` kwargs.
- `CMY` no longer accepts `observer`, `illuminant` or `rgb_type` kwargs.
- `CMYK` no longer accepts `observer`, `illuminant` or `rgb_type` kwargs.
- Removed 'debug' kwargs in favor of Python's logging.
- Completely re-worked exception list. Eliminated some redundant exceptions, re-named basically everything else.

#### Features

- Python 3.3 support added.
- Added `tox.ini` with supported environments.
- Removed the old custom test runner in favor of nose.
- Replacing simplified RGB->XYZ conversions with Bruce Lindbloom's.
- A round of PEP8 work.
- Added a `BaseColorConversionTest` test class with some greatly improved color comparison. Much more useful in tracking down breakages.
- Eliminated non-matrix delta E computations in favor of the matrix equivalents. No need to maintain duplicate code, and the matrix stuff is faster for bulk operations.

## Bugs

- Corrected delta\_e CMC example error. Should now run correctly.
- color\_diff\_matrix.delta\_e\_cie2000 had an edge case where certain angles would result in an incorrect delta E.
- Un-hardcoded XYZColor.apply\_adaptation()'s adaptation and observer angles.

## 2.7.2 1.0.9

### Features

- Added an optional vectorized deltaE function. This uses NumPy array/matrix math for a very large speed boost. (Eddie Bell)
- Added this changelog.

## Bugs

- Un-hardcode the observer angle in adaptation matrix. (Bastien Dejean)

## 2.7.3 1.0.8

- Initial GitHub release.



---

## Useful color math resources

---

- [Bruce Lindbloom](#) - Lots of formulas, calculators, and standards.
- [John the Math Guy](#) - Useful tutorials and explanations of color theory.