python-colorspace Documentation

Release 0.0.1

Reto Stauffer

Oct 08, 2018

Contents

`python-colorspace` is a python package to create and handle colors and color palettes in python. Based on the Hue-Chroma-Luminance (HCL) color space effective color palettes can be designed and implemented in your own daily workflow.

This package is based on the code and ideas of the R colorspace package as it has often been requested by python enthusiasts. More information and an interactive interface can also be found on HCLwizard.org.

The package itself can be found on github this documentation is also available on ReadTheDocs.

CHAPTER 1

Contents

1.1 Getting started

1.1.1 Installation

The package is available on github) and can thus simply be installed via `pip`. The package is tested against Python *2.7* and Python *3.6+*. The only dependency is numpy, for some methods the matplotlib might be required.

- `pip install git+https://github.com/retostauffer/python-colorspace`,
- Or clone the repository and use the good old `python setup.py install`.

Some of the functions (those creating plots and manipulating images) depend on matplotlib and imageio. The package will raise an error and inform you about the dependency as soon as you run into them. However, depending on what want to do there is no need for these two additional modules.

After successfully installing the package you might want to have a look into our *Getting started* section of this documentation.

1.1.2 The HCL Color Space

The most well known color space is the Red-Green-Blue (RGB) color space. The RGB color space is based the technical demand of digital screens such as computer monitors or TVs. The color of each individual pixel on a screen is created by mixing intensities of red, green, and blue (additive color mixture) to create the picture we can see.

In contrast, the Hue-Chroma-Luminance color space is based on how human color perception works. In contrast to computer screens our visual system (eye-brain) is processing visual information in the dimension of luminance (the lightness of an object), chroma (the colorfulness) and hue (the actual color tone).

The Hue-Chroma-Luminance color space allows us to directly control each of these three dimensions. The swatch plot below illustrates how this works. Each of the three swatches varies only one specific dimension (hue, chroma, and luminance) while all others are held constant. Thus, all colors in the top swatch exhibit the same lightness and same color intensity while only the hue changes from left to right from reddish over greenish, blueish, back to red. The second swatch has constant luminance and hue, but varies from zero chroma (pure gray) on the left hand side to a

vivid red. The last swatch goes from black (zero luminance) to white (full luminance) with zero chroma which yields pure gray scale colors.

Path trough the HCL space

The properties of the HCL color space allows to draw well-defined and effective color palettes from the HCL color space. One example are the *Diverging HCL palettes*. The design principle of diverging color palettes is that both ends of the spectrum have the same luminance and chroma to not distort the data and add artificial weight to one side or the other. The only property which changes from one end to the other is the hue (e.g., from red to blue).

The animation below shows the HCL color space as a volume and the path of the default *Diverging HCL palettes* ("Blue-Red") trough the space. The vertical axis shows the luminance dimension from `L=0` (black) to `L=100` (white), hue and chroma are shwon on the XY plane. The angle (from `H=0` to `H=360`; cyclic) defines the hue, the radial distance to the center the chroma.

The solid line inside the volume shows the path of the "Blue-Red" color palette with 11 unique colors. Both sides of the palette proceed linearly from a neutral center point with 90% luminance (`L=90`) and no chroma to a dark blue (`H=260`; constant) and a dark red (`H=0`; constant) with equal luminance (`L=30`) and equal chroma (`C=80`).

This yields a well balanced diverging color map shown in the top left corner of the animation with equal weights on both ends which can easily be adjusted by adjusting the start and end point of the path without and preserve the well defined and monotonic behavior of the overall palette. Variations of the "Blue-Red" diverging color map can be found on the "*default color palette page*".

Why is the Luminance Important

The following non-technical example illustrates why it is beneficial to have direct control over the luminance dimension. The image below shows a juicy and delicious apple. Our visual system needs only the blink of an eye to identify the object as what it is.

This simple example illustrates how false or missing luminance information can quickly obscure the information of a figure or graph. Even with additional attributes on top (hue/chroma) the underlying luminance information is processed by our visual system and helps us to gather the information we are looking at. If used in a wrong way, colors can easily wreck the effectiveness of a (scientific) visualization and might, in a worst case, even mislead the reader in a way that he is perceiving something else or at least focussing on the wrong aspects of the image.

And here the python-colorspace package can help you out with designing effective color maps, investigate the properties of a new or existing color palette, and more.

Effective Color Palettes

Todo: Introduction to the three basic principles of the diverging, sequential, and qualitative color maps.

1.1.3 Usage Examples

Precipitation Forecast

1.1.4 Matolotlib cmaps

For demonstration the 3D surface demo is used to demonstrate the colorspace cmap functionality. The code for the demo can be *a the end of this page*.

HCL Color Palettes

All *palettes.hclpalette* objects provide a method called *palettes.hclpalette.cmap()* method which returns a matplotlib color map with `n` colors (default `51`).

The example below shows the demo with the "Green-Orange" *palettes.diverging_hcl* color palette and the "Purple-Orange" *palettes.sequential_hcl* color palette in the top row, and the "Set 2" *palettes.qualitative_hcl* and a matplotlib default color map called `gist_ncar` in the bottom row.

Please note: that none of the two shown in the bottom row should be used to illustrate such a data set. The *palettes.qualitative_hcl* color map has iso-chroma (constant color intensity) and iso-luminance (constant lightness) and only varies in the hue dimension (the color itself). Such palettes are made for classification tasks and not to display a data set as shown in the demo. The `gist_ncar` palette should also not be used due to the immense discontinuity across the palette (I've chosen this as one of the worst examples among the matplotlib color maps). More information about *effective color palettes* can be found *on this page*.

Color Vision Deficiency

The color vision deficiency (CVD) toolbox of the colorspace package also allows to simulate color vision deficiencies on `matplotlib.colors.LinearSegmentedColormap` color maps.

The figure above shows the very same color map (the "Blue-Red" default *palettes.diverging_hcl* palette) in four different versions. Top left is the original color map as people without visual constraints perceive the colors. Top right is a desaturated version where all the color information is removed. This yields a pure gray scale color map, and as the diverging color maps are well balanced, to equal gray on both ends of the spectrum.

The bottom row shows how people with a deuteranomaly (commonly known as "red-green blindness"; left) and protanomaly (less common, known as "blue-yellow blindness"; right) perceive the very same color map. Except for the desaturated version the color map works quite well and even under visual constraints the association between color and the actual value is possible without any problem. For other color maps this might not be true and the effectiveness of a color map can rapidly collapse under certain visual constraints. Thus, always think of who receives the figures and graphs, and whether or not it is important that color vision deficiency has to be considered or not (most often the answer is: yes!).

The Demo Function

The output below shows the `demo` function used on this page. It is a modified version of the 3D surface example.

```
def demo(*args):
    """demo(*args)

    3D surface (color map) example from matplotlib.org

    Parameters
```

(continues on next page)

(continued from previous page)

```
    ----------
    args : ...
        a set of LinearSegmentedColormap our custom
        matplotlib.color.LinearSegmentedColormap.
    """

    from mpl_toolkits.mplot3d import Axes3D
    import matplotlib.pyplot as plt
    from matplotlib.ticker import LinearLocator, FormatStrFormatter
    import numpy as np

    # Subplot config
    nsubplots = len(args)
    fig = plt.figure()

    # Make data.
    X    = np.arange(-5, 5, 0.01)
    Y    = np.arange(-5, 5, 0.01)
    X, Y = np.meshgrid(X, Y)
    R    = np.sqrt(X**2 + Y**2)
    Z    = np.sin(R)

    # Disable axis
    def disable_axis(ax):
        for a in (ax.w_xaxis, ax.w_yaxis, ax.w_zaxis):
            for t in a.get_ticklines()+a.get_ticklabels():
                t.set_visible(False)
            a.pane.set_visible(False)

    # Plotting surface(s)
    for i,cmap in enumerate(args):

        # Plot the surface.
        ax   = fig.add_subplot(np.ceil(nsubplots/2.), 2, i+1, projection='3d')
        surf = ax.plot_surface(X, Y, Z, cmap = cmap,
                               linewidth=0, antialiased=False)

        # Customize the z axis.
        ax.set_title(cmap.name)
        ax.set_zlim(-1.01, 1.01)
        disable_axis(ax)
        ax.zaxis.set_major_locator(LinearLocator(10))
        ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

        # Add a color bar which maps values to colors.
        fig.colorbar(surf, shrink=0.5, aspect=5)
        ax.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)

    fig.tight_layout()
    plt.show()
```

1.1.5 In Contrast to R

This package is inspired and based on the *R* colorspace package. The two packages are very similar, however, some of the objects and methods differ to allow to use the color maps in python in an easy and "native" way.

1.1.6 Release Notes

Availability

The software is published under the Gnu Public License on github. Please feel free to use, share, improve, and redistribute the software as long as the attribution is given correctly.

Please also feel free to help improving the software (via pull requests, or simply get in touch with me, helping hands would be very welcome)!

Version 0.1.0 (September 17, 2018)

Early beta release.

A wide range of methods are already implemented and roughly tested. I've decided to launch it as an early beta release to get some feedback from those who use python more frequently than I do! Feel free to report bugs, ideas, or even contribute.

I'll try to update update the documentation as soon as possible and to improve the package itself, if I can find a free time slot.

Version 0.0.1 (beginning of September 2018)

Development version.

First implementation of the `colorspace` package in python. This is still an early alpha version, I am currently working on better documentation, testing, and getting the necessary classes and objects into the package to provide a useful toolbox for python enthusiasts.

1.1.7 Checks against R

1.2 Examples

1.2.1 Selecting Color Palettes

There are different ways to access color palettes.

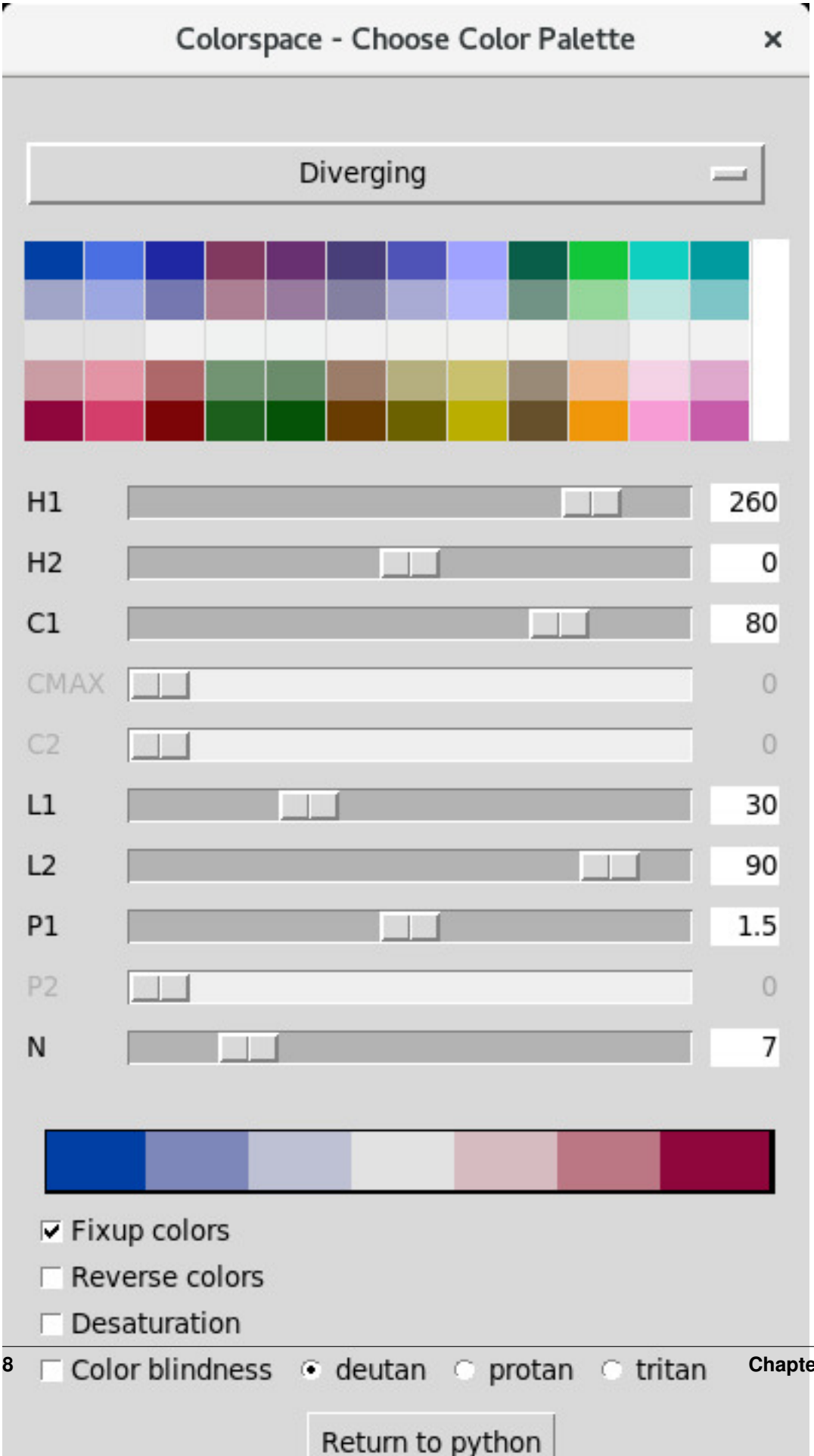
Default Color Palettes

The package provides a set of effective and well selected color palettes which can directly be accessed by name.

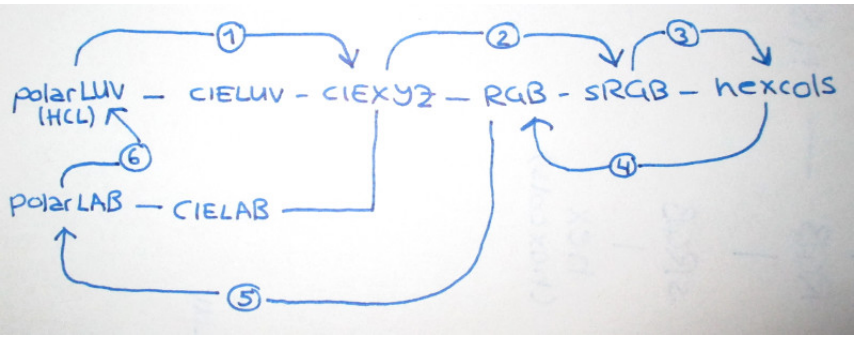
- More information: *Graphical User Interface*

Graphical User Interface

`choose_palette()`.


Colorspace - Choose Color Palette
Diverging
H1
260
H2
0
C1
80
CMAX
0
C2
0
L1
30
L2
90
P1
1.5
P2
0
N
7
Fixup colors
Reverse colors
Desaturation
Color blindness
deutan
protan
tritan
Return to python

1.2.2 Color transformation



1.2.3 Color Vision Deficiency Emulator

A function called *cvd_emulator* allows to . . .

Function Reference

`cvd_emulator.`**`cvd_emulator`**(*image = "DEMO", cvd = "desaturate", severity = 1.0, output = None, dropalpha = False*)

Simulate color deficiencies on png/jpg/jpeg figures. Takes an existing pixel image and simulates different color vision deficiencies.

The function displays a matplotlib figure if output is set to None. If the parameter output is set, the converted figure will be stored. If only one color vision deficiency is defined (e.g., `cvd = "desaturate"`) a figure of the same type and size as the input figure will be saved to the disc. If multiple cvd's are specified a multi-panel plot will be stored under *output*.

Parameters

- **`image`** (*`str`*) – name of the figure which should be converted (png/jpg/jpeg). If `image = "DEMO"` the package demo figure will be used.
- **`cvd`** (*`str, list`*) – the color vision deficiency. Allowed types are deutanope, protanope, tritanope, and desaturated. Input is either a single string or a list of strings which define the cvd's which should be simulated.
- **`severity`** (*`float`*) – how severe the color vision deficiency is (`[0.,1.]`). Also used as the amount of desaturation if `cvd = "desaturate"`.
- **`output`** (*`string`*) – optional. It None an interactive plotting window will be opened. If a string is given the figure will be written to `output`.
- **`dropalpha`** (*`bool`*) – whether or not to drop the alpha channel. Only useful for figures having an alpha channel (png w/ alpha).

Examples

```
>>> from colorspace.cvd_emulator import cvd_emulator
>>> cvd_emulator("DEMO", "deutan", 0.5)
>>> cvd_emulator("DEMO", "desaturate", 1.0, "output.png")
>>> cvd_emulator("DEMO", ["original", "deutan", "protan"], 0.5, dropalpha = True)
```

Note: Requires the modules `matplotlib` and `imageio`.

1.3 References

1.3.1 Default Palettes

Diverging HCL palettes

Usage information: the names of the palettes can be used to retrieve a set of colors by calling:

Function Reference

class palettes.**diverging_hcl**(*h = [260, 0], c = 80, l = [30, 90], power = 1.5, fixup = True, palette = None, rev = False, *args, **kwargs*)

Diverging HCL color palette.

Parameters

- **h** (*numeric list*) – hue values, diverging color palettes should have different hues for both ends of the palette. If only one value is present it will be recycled ending up in a diverging color palette with the same colors on both ends. If more than two values are provided the first two will be used while the rest is ignored. If input `h` is a string this argument acts like the `palette` argument (see `palette` input parameter).
- **c** (*numeric*) – chroma value, a single numeric value. If multiple values are provided only the first one will be used.
- **l** (*numeric list*) – luminance values. The first value is for the two ends of the color palette, the second one for the neutral center point. If only one value is given this value will be recycled.
- **power** (*numeric*) – power parameter for non-linear behaviour of the color palette.
- **fixup** (*bool*) – only used when converting the HCL colors to hex. Should RGB values outside the defined RGB color space be corrected?
- **palette** (*string*) – can be used to load a default diverging color palette specification. If the palette does not exist an exception will be raised. Else the settings of the palette as defined will be used to create the color palette.
- **rev** (*bool*) – should the color map be reversed.
- **args** – unused.
- **kwargs** – Additional arguments to overwrite the h/c/l settings. @TODO has to be documented.

Returns

- *Initialize new object, no return. Raises a set of errors if the parameters*
- *are misspecified. Note that the object is callable, the default object call*
- can be used to return hex colors (identical to the `.colors()` method),
- *see examples.*

Examples

```
>>> from colorspace import diverging_hcl
>>> a = diverging_hcl()
>>> a.colors(10)
>>> b = diverging_hcl("Blue-Yellow 3")
>>> b.colors(10)
>>> # The standard call of the object also returns hex colors. Thus,
>>> # you can make your code slimmer by calling:
>>> diverging_hcl("Dynamic")(10)
```

colors(*n = 11*, *type_ = "hex"*, *fixup = None*)

Returns the colors of the current color palette.

Parameters

- **n** (*int*) – number of colors which should be returned.
- **fixup** (*None, bool*) – should sRGB colors be corrected if they lie outside the defined color space? If `None` the `fixup` parameter from the object will be used. Can be set to `True` or `False` to explicitly control the fixup here.
- **alpha** (*None, float*) – float (single value) or vector of floats in the range of `[0., 1.]` for alpha transparency channel (`0.` means full transparency, `1.` opaque). If a single value is provided it will be applied to all colors, if a vector is given the length has to be `n`.

Sequential

- `colorspace.sequential_hcl(n = 10, name = "<palette name>")`

Function Reference

class palettes.**sequential_hcl**(*h = 260, c = [80, 30], l = [30, 90], power = 1.5, fixup = True, palette = None, rev = False, *args, **kwargs*)

Sequential HCL color palette.

Parameters

- **h** (*numeric*) – hue values. If only one value is given the value is recycled which yields a single-hue sequential color palette. If input *h* is a string this argument acts like the *palette* argument (see *palette* input parameter).
- **c** (*numeric list*) – chroma values, numeric of length two. If multiple values are provided only the first one will be used.
- **l** (*numeric list*) – luminance values, numeric of length two. If multiple values are provided only the first one will be used.
- **power** (*numeric, numeric list*) – power parameter for non-linear behaviour of the color palette. One or two values can be provided.
- **fixup** (*bool*) – only used when converting the HCL colors to hex. Should RGB values outside the defined RGB color space be corrected?
- **palette** (*string*) – can be used to load a default diverging color palette specification. If the palette does not exist an exception will be raised. Else the settings of the palette as defined will be used to create the color palette.

- **rev** (*bool*) – should the color map be reversed.
- **args** – unused.
- **kwargs** – Additional arguments to overwrite the h/c/l settings. @TODO has to be documented.

Returns

- *Initialize new object, no return. Raises a set of errors if the parameters*
- *are misspecified. Note that the object is callable, the default object call*
- can be used to return hex colors (identical to the `.colors()` method),
- *see examples.*

Examples

```
>>> from colorspace import sequential_hcl
>>> a = sequential_hcl()
>>> a.colors(10)
>>> b = sequential_hcl("Reds")
>>> b.colors(10)
>>> # The standard call of the object also returns hex colors. Thus,
>>> # you can make your code slimmer by calling:
>>> sequential_hcl("Dynamic")(10)
```

colors(*n = 11*, *type_ = "hex"*, *fixup = None*)

Returns the colors of the current color palette.

Parameters

- **n** (*int*) – number of colors which should be returned.
- **fixup** (*None, bool*) – should sRGB colors be corrected if they lie outside the defined color space? If `None` the `fixup` parameter from the object will be used. Can be set to `True` or `False` to explicitly control the fixup here.

Qualitative

Function Reference

class palettes.**qualitative_hcl**(*h = [0, 360.]*, *c = 50*, *l = 70*, *fixup = True*, *palette = None*, *rev = False*, ***kwargs*)

Qualitative HCL color palette.

Parameters

- **h** (*numeric list*) – hue values, qualitative color palettes require two hues. If more than two values are provided the first two will be used while the rest is ignored. If input *h* is a string this argument acts like the *palette* argument (see *palette* input parameter).
- **c** (*numeric*) – chroma value, a single numeric value. If multiple values are provided only the first one will be used.
- **l** (*numeric*) – luminance value, a single numeric value. If multiple values are provided only the first one will be used.

- **fixup** (*bool*) – only used when converting the HCL colors to hex. Should RGB values outside the defined RGB color space be corrected?
- **palette** (*None, string*) – can be used to load a default diverging color palette specification. If the palette does not exist an exception will be raised. Else the settings of the palette as defined will be used to create the color palette.
- **rev** (*bool*) – should the color map be reversed.
- **args** – unused.
- **kwargs** – Additional arguments to overwrite the h/c/l settings. @TODO has to be documented.

Returns

- *Initialize new object, no return. Raises a set of errors if the parameters*
- *are misspecified. Note that the object is callable, the default object call*
- can be used to return hex colors (identical to the `.colors()` method),
- *see examples.*

Examples

```
>>> from colorspace import diverging_hcl
>>> a = qualitative_hcl()
>>> a.colors(10)
>>> b = qualitative_hcl("Dynamic")
>>> b.colors(10)
>>> # The standard call of the object also returns hex colors. Thus,
>>> # you can make your code slimmer by calling:
>>> qualitative_hcl("Dynamic")(10)
```

colors(*n = 11*, *type_ = "hex"*, *fixup = None*)

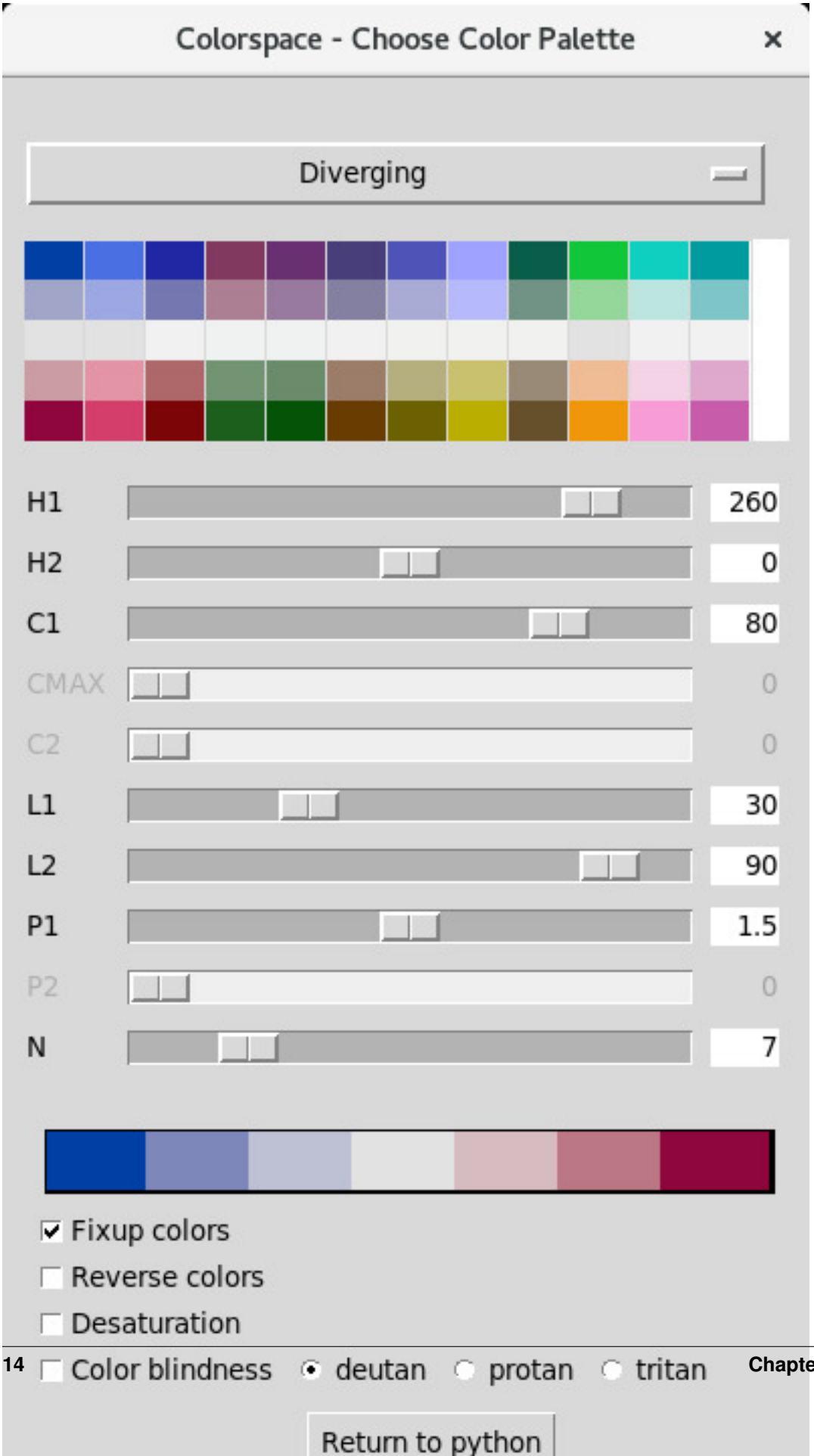
Returns the colors of the current color palette.

Parameters

- **n** (*int*) – number of colors which should be returned.
- **fixup** (*None, bool*) – should sRGB colors be corrected if they lie outside the defined color space? If `None` the `fixup` parameter from the object will be used. Can be set to `True` or `False` to explicitly control the fixup here.

1.3.2 Graphical User Interface

`choose_palette()`.


Colorspace - Choose Color Palette
Diverging
H1
260
H2
0
C1
80
CMAX
0
C2
0
L1
30
L2
90
P1
1.5
P2
0
N
7
Fixup colors
Reverse colors
Desaturation
Color blindness
deutan
protan
tritan
Return to python

Function Reference

choose_palette.**choose_palette**(***kwargs*)

Graphical user interface to choose HCL based color palettes. Returns an object of *palettes.diverging_hcl*, *palettes.qualitative_hcl*, or *palettes.sequential_hcl* with user-defined default settings.

Parameters **kwargs** – See *choose_palette.gui*.

Returns The object allows to get colors in different ways, the default is a list with hex colors. See *palettes.hclpalette* or, more specifically, the manual of the depending palette (*palettes.diverging_hcl*, *palettes.qualitative_hcl*, or *palettes.sequential_hcl*).

Return type *palettes.hclpalette* object

1.3.3 User API

This page contains the basic API methods and objects you might need when starting with the `python-colorspace` package.

More detailed descriptions of the different classes and methods can be found here:

- ...
- ...
- ...
- ...

Main Object Types

Palette Functions

Graphical User Interface

1.3.4 Examples of Default Palettes

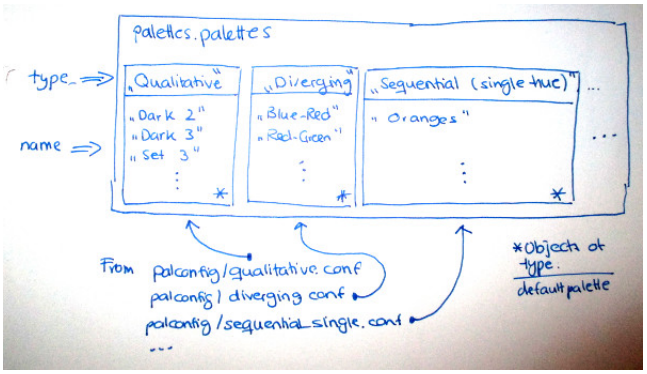
The *colorspace* package contains a set of default palettes which can be accessed via palette name.

Available Palettes

1.4 Developer

1.4.1 Palettes

The *palettes* module provides a user-friendly interface to HCL based color palettes.



Rainbow HCL

class palettes.**rainbow_hcl**(*c = 50*, *l = 70*, *start = 0*, *end = 360*, *gamma = None*, *fixup = True*, *rev = False*, **args*, ***kwargs*)

HCL rainbow, a qualitative cyclic rainbow color palette with uniform luminance and chroma.

Parameters

- **c** (*int*) – chroma of the color map `[0-100+]`.
- **l** (*int*) – luminance of the color map `[0-100]`.
- **start** (*int*) – hue at which the rainbow should start.
- **end** (*int*) – hue at which the rainbow should end.
- **rev** (*bool*) – should the color map be reversed.
- **gamma** (*float*) – gamma value used for transfiromation from/to sRGB. @TODO implemented? Check!
- **fixup** (*bool*) – only used when converting the HCL colors to hex. Should RGB values outside the defined RGB color space be corrected?

Returns

- *Initialize new object, no return. Raises a set of errors if the parameters*
- *are misspecified. Note that the object is callable, the default object call*
- can be used to return hex colors (identical to the `.colors()` method),
- *see examples.*

Example

```
>>> from colorspace import rainbow_hcl
>>> pal = rainbow_hcl()
>>> pal.colors(3); pal.colors(20)
>>> # The standard call of the object also returns hex colors. Thus,
>>> # you can make your code slimmer by calling:
>>> rainbow_hcl("Dynamic")(10)
```

Qualitative HCL

class palettes.**qualitative_hcl**(*h = [0, 360.], c = 50, l = 70, fixup = True, palette = None, rev = False, **kwargs*)

Qualitative HCL color palette.

Parameters

- **h** (*numeric list*) – hue values, qualitative color palettes require two hues. If more than two values are provided the first two will be used while the rest is ignored. If input *h* is a string this argument acts like the *palette* argument (see *palette* input parameter).
- **c** (*numeric*) – chroma value, a single numeric value. If multiple values are provided only the first one will be used.
- **l** (*numeric*) – luminance value, a single numeric value. If multiple values are provided only the first one will be used.
- **fixup** (*bool*) – only used when converting the HCL colors to hex. Should RGB values outside the defined RGB color space be corrected?
- **palette** (*None, string*) – can be used to load a default diverging color palette specification. If the palette does not exist an exception will be raised. Else the settings of the palette as defined will be used to create the color palette.
- **rev** (*bool*) – should the color map be reversed.
- **args** – unused.
- **kwargs** – Additional arguments to overwrite the h/c/l settings. @TODO has to be documented.

Returns

- *Initialize new object, no return. Raises a set of errors if the parameters*
- *are misspecified. Note that the object is callable, the default object call*
- can be used to return hex colors (identical to the `.colors()` method),
- *see examples.*

Examples

```
>>> from colorspace import diverging_hcl
>>> a = qualitative_hcl()
>>> a.colors(10)
>>> b = qualitative_hcl("Dynamic")
>>> b.colors(10)
>>> # The standard call of the object also returns hex colors. Thus,
>>> # you can make your code slimmer by calling:
>>> qualitative_hcl("Dynamic")(10)
```

colors(*n = 11, type_ = "hex", fixup = None*)

Returns the colors of the current color palette.

Parameters

- **n** (*int*) – number of colors which should be returned.

- **fixup** (*None, bool*) – should sRGB colors be corrected if they lie outside the defined color space? If `None` the `fixup` parameter from the object will be used. Can be set to `True` or `False` to explicitly control the fixup here.

Diverging HCL

class palettes.**diverging_hcl**(*h = [260, 0], c = 80, l = [30, 90], power = 1.5, fixup = True, palette = None, rev = False, *args, **kwargs*)

Diverging HCL color palette.

Parameters

- **h** (*numeric list*) – hue values, diverging color palettes should have different hues for both ends of the palette. If only one value is present it will be recycled ending up in a diverging color palette with the same colors on both ends. If more than two values are provided the first two will be used while the rest is ignored. If input `h` is a string this argument acts like the `palette` argument (see `palette` input parameter).
- **c** (*numeric*) – chroma value, a single numeric value. If multiple values are provided only the first one will be used.
- **l** (*numeric list*) – luminance values. The first value is for the two ends of the color palette, the second one for the neutral center point. If only one value is given this value will be recycled.
- **power** (*numeric*) – power parameter for non-linear behaviour of the color palette.
- **fixup** (*bool*) – only used when converting the HCL colors to hex. Should RGB values outside the defined RGB color space be corrected?
- **palette** (*string*) – can be used to load a default diverging color palette specification. If the palette does not exist an exception will be raised. Else the settings of the palette as defined will be used to create the color palette.
- **rev** (*bool*) – should the color map be reversed.
- **args** – unused.
- **kwargs** – Additional arguments to overwrite the h/c/l settings. @TODO has to be documented.

Returns

- *Initialize new object, no return. Raises a set of errors if the parameters*
- *are misspecified. Note that the object is callable, the default object call*
- can be used to return hex colors (identical to the `.colors()` method),
- *see examples.*

Examples

```
>>> from colorspace import diverging_hcl
>>> a = diverging_hcl()
>>> a.colors(10)
>>> b = diverging_hcl("Blue-Yellow 3")
>>> b.colors(10)
>>> # The standard call of the object also returns hex colors. Thus,
```

(continues on next page)

(continued from previous page)

```
>>> # you can make your code slimmer by calling:
>>> diverging_hcl("Dynamic")(10)
```

colors (*n = 11*, *type_ = "hex"*, *fixup = None*)

Returns the colors of the current color palette.

Parameters

- **n** (*int*) – number of colors which should be returned.
- **fixup** (*None, bool*) – should sRGB colors be corrected if they lie outside the defined color space? If `None` the `fixup` parameter from the object will be used. Can be set to `True` or `False` to explicitly control the fixup here.
- **alpha** (*None, float*) – float (single value) or vector of floats in the range of `[0., 1.]` for alpha transparency channel (`0.` means full transparency, `1.` opaque). If a single value is provided it will be applied to all colors, if a vector is given the length has to be `n`.

Sequential HCL

class palettes.**sequential_hcl**(*h = 260, c = [80, 30], l = [30, 90], power = 1.5, fixup = True, palette = None, rev = False, *args, **kwargs*)

Sequential HCL color palette.

Parameters

- **h** (*numeric*) – hue values. If only one value is given the value is recycled which yields a single-hue sequential color palette. If input *h* is a string this argument acts like the *palette* argument (see *palette* input parameter).
- **c** (*numeric list*) – chroma values, numeric of length two. If multiple values are provided only the first one will be used.
- **l** (*numeric list*) – luminance values, numeric of length two. If multiple values are provided only the first one will be used.
- **power** (*numeric, numeric list*) – power parameter for non-linear behaviour of the color palette. One or two values can be provided.
- **fixup** (*bool*) – only used when converting the HCL colors to hex. Should RGB values outside the defined RGB color space be corrected?
- **palette** (*string*) – can be used to load a default diverging color palette specification. If the palette does not exist an exception will be raised. Else the settings of the palette as defined will be used to create the color palette.
- **rev** (*bool*) – should the color map be reversed.
- **args** – unused.
- **kwargs** – Additional arguments to overwrite the h/c/l settings. @TODO has to be documented.

Returns

- *Initialize new object, no return. Raises a set of errors if the parameters*
- *are misspecified. Note that the object is callable, the default object call*
- can be used to return hex colors (identical to the `.colors()` method),
- *see examples.*

Examples

```
>>> from colorspace import sequential_hcl
>>> a = sequential_hcl()
>>> a.colors(10)
>>> b = sequential_hcl("Reds")
>>> b.colors(10)
>>> # The standard call of the object also returns hex colors. Thus,
>>> # you can make your code slimmer by calling:
>>> sequential_hcl("Dynamic")(10)
```

colors(*n = 11*, *type_ = "hex"*, *fixup = None*)

Returns the colors of the current color palette.

Parameters

- **n** (*int*) – number of colors which should be returned.
- **fixup** (*None, bool*) – should sRGB colors be corrected if they lie outside the defined color space? If `None` the `fixup` parameter from the object will be used. Can be set to `True` or `False` to explicitly control the fixup here.

HCL Palette Baseclass

The different HCL color palettes (*palettes.rainbow_hcl*, *palettes.diverging_hcl*, *palettes.qualitative_hcl*) are extending tihs *palettes.hclpalette* base class.

The *palettes.hclpalette* class provides several methods to interact with the HCL palette objects above to e.g., extract colors or check the current palette settings.

class palettes.**hclpalette**

Hy, I am the base class. Is extended by the different HCL based color palettes such as the classes diverging_hcl, qualitative_hcl, rainbow_hcl, sequential_hcl, and maybe more in the future.

cmap(*n = 51*, *name = "custom_hcl_cmap"*)

Allows to retrieve a matplotlib LinearSegmentedColormap color map. Clasically LinearSegmentedColormaps allow to retrieve a set of `N` colors from a set of `n` colors where `N >> n`. The matplotlib simply linearely interpolates between all `n` colors to extend the number of colors to `N`.

In case of *hclpalette()* objects this is not necessary as *hclpalette()* objects allow to retrieve `N` colors directly along well-specified Hue-Chroma-Luminance paths. Thus, this method returns a matplotlib color map with `n==N` colors. The linear interpolation between the colors (as typically done by LinearSegmentedColormap) is not necessary. However, for convenience cmaps have been implemented such that you can easily use hcl based palettes in your existing workflow.

Parameters

- **n** (*int*) – number of colors
- **name** (*str*) – name of the custom color map. Default is `custom_hcl_cmap`

Returns Returns a `LinearSegmentedColormap` (cmap) to be used with the matplotlib library.

Return type matplotlib.colors.LinearSegmentedColormap

get(*key*)

Returns one specific item of the palette settings, e.g., the current value for `h1` or `l2`. If not existing a *None* will be returned.

Parameters **key** (*str*) – name of the setting to be returned.

Returns

- None if `key` does ont exist, else the current value will be
- *returned.*

Examples

```
>>> from colorspace.palettes import rainbow_hcl
>>> a = rainbow_hcl()
>>> a.get("h1")
>>> a.get("c1")
>>> a.get("l1")
>>> a.get("not_defined")
```

name()

Returns

Return type Returns the name of the palette, string.

show_settings()

Shows the current settings (table like print to stdout). Should more be seen as a development method than a very useful thing.

Examples

```
>>> from colorspace.palettes import rainbow_hcl
>>> a = rainbow_hcl(10)
>>> a.show_settings()
```

specplot(*n = 180*, **args*, ***kwargs*)

Interfacing the *specplot.specplot()* function. Plotting the spectrum of the current color palette.

Parameters

- **n** (*int*) – number of colors.
- **args** – forwarded to *specplot.specplot()*.
- **kwargs** – forwarded to *specplot.specplot()*.

Examples

```
>>> from colorspace import diverging_hcl
>>> pal = diverging_hcl()
>>> pal.specplot()
>>> pal.specplot(rgb = False)
```

swatchplot(*n = 7*)

Interfacing the `swatchplot.swatchplot()` function. Plotting the spectrum of the current color palette.

Parameters **n** (*int*) – number of colors.

Examples

```
>>> from colorspace import diverging_hcl
>>> pal = diverging_hcl()
>>> pal.swatchplot()
>>> pal.swatchplot(n = 21)
```

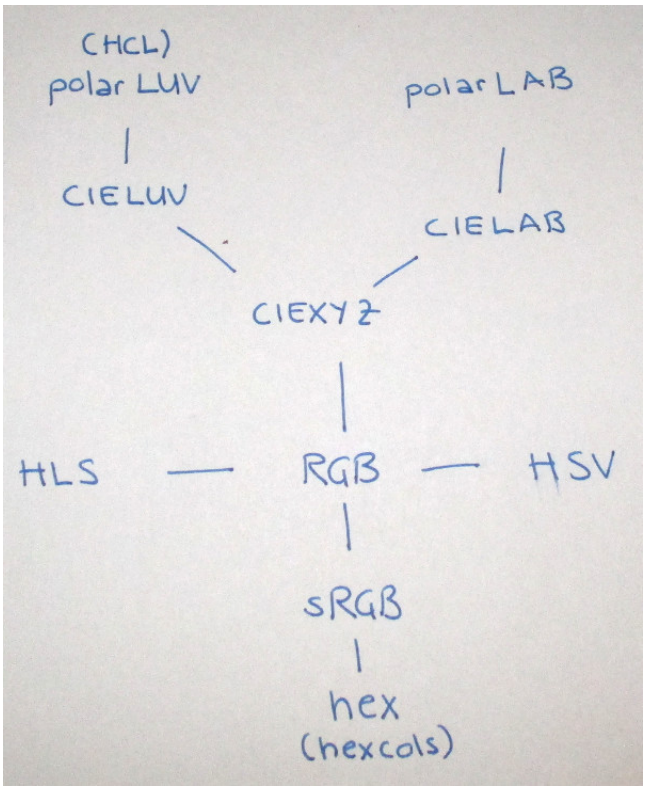
1.4.2 Other Methods

class palettes.**palette**(*colors*, *name*)
Custom named color palette with a fixed number of colors. Used for `hcl_palettes.swatchplot()`.

class palettes.**hclpalette**
Hy, I am the base class. Is extended by the different HCL based color palettes such as the classes diverging_hcl, qualitative_hcl, rainbow_hcl, sequential_hcl, and maybe more in the future.

1.4.3 The color objects

The `colorlib` module is handling the transformation between different color spaces. A set of different color spaces is available including *colorlib.CIELUV*, *colorlib.CIELAB*, *colorlib.CIEXYZ*, *colorlib.polarLUV*/*colorlib.HCL*, *colorlib.polarLAB*, *colorlib.HSV*, *colorlib.HLS*, *colorlib.RGB*, *colorlib.sRGB*, and *colorlib.hexcols* for HEX colors.



class colorlib.**colorobject**
This is the base class of all color objects and provides some default methods.

colors (*fixup = True*)

Returns hex colors of the current *colorobject*. Converts the colors into a *hexcols* object to retrieve hex colors which are returned as list.

If the object contains alpha values the alpha level is added to the hex string if and only if alpha is not equal to 1.0.

Parameters

- **fixup** (*bool*) – whether or not to correct rgb values outside the defined range of `[0., 1.]`
- **rev** (*bool*) – return colors in reversed order?

Returns Returns a list of hex colors.

Return type list

Examples

```
>>> from colorspace.colorlib import HCL
>>> cols = HCL([0, 40, 80], [30, 60, 80], [85, 60, 35])
>>> cols.colors()
```

dropalpha()

Remove alpha information from the color object, if defined.

get (*dimname = None*)

Returns the current color coordinates. Either a single coordinate (if `dimname` is set) or all coordinates of the current *colorobject*. The latter will return a dictionary containing the data with all defined dimensions.

Parameters **dimname** (*None, str*) – can be set to only retrieve one very specific coordinate.

Returns Either a dictionary or a single numpy.ndarray if input `dimname` was not specified. If a specific dimension is requested bu the dimension does not exist a ValueError is raised.

Return type dict or numpy.ndarray

Examples

```
>>> from colorspace.colorlib import HCL
>>> cols = HCL([260, 80, 30], [80, 0, 80], [30, 90, 30])
>>> cols.get()
>>> cols.get("H")
```

get_whitepoint()

A white point definition is used to adjust the colors. If not explicitly set via *set_whitepoint()* default values are used. This method returns the definition of the white point in use.

Returns Returns a dict with *X*, *Y*, *Z*, the white point specification for the three dimensions.

Return type dict

Examples

```
>>> from colorspace.colorlib import hexcols
>>> c = hexcols("#ff0000")
>>> c.get_whitepoint()
```

hasalpha()
Small helper function to check whether the current color object has alpha values or not.

Returns Returns `True` if alpha values are present, `False` if not.

Return type bool

set(*kwargs*)
Allows to manipulate the current colors. The named input arguments have to fulfil a specific set or requirements. If not, the function raises a ValueError.

The dimension has to exist, and the new data have to be of the same type and of the same length to be accepted.

No return, modifies the current object.

Parameters **kwargs** – named arguments. The key is the name of the dimension to be changed, the value an object which fulfills the requirements (see description of this method)

Examples

```
>>> from colorspace.colorlib import HCL
>>> cols = HCL([260, 80, 30], [80, 0, 80], [30, 90, 30])
>>> print cols
>>> cols.set(H = [150, 150, 30])
>>> print cols
```

set_whitepoint(***kwargs*)
A white point definition is used to adjust the colors. This method allows to set custom values. If not explicitly set a default specification is used.

No return, stores the new definition on the object. *get_whitepoint()* can be used to get the current specification.

Parameters

- **X** (*float*) – white specification for dimension X
- **Y** (*float*) – white specification for dimension Y
- **Z** (*float*) – white specification for dimension Z

Examples

```
>>> from colorspace.colorlib import hexcols
>>> c = hexcols("#ff0000")
>>> c.set_whitepoint(X = 100., Y = 100., Z = 101.)
>>> c.get_whitepoint()
```

specplot(***kwargs*)
Plotting a specplot (see *specplot.specplot()*) of the current color object. Additional arguments can be used to control the specplot.

Parameters **kwargs** – named list of additional arguments forwarded to the specplot function

Examples

```
>>> from colorspace.colorlib import HCL
>>> cols = HCL([260, 80, 30], [80, 0, 80], [30, 90, 30])
>>> cols.specplot()
>>> cols.specplot(rgb = False)
```

swatchplot(*n = 7*)
Interfacing the `swatchplot.swatchplot()` function. Plotting the spectrum of the current color palette.

Examples

```
>>> from colorspace.colorlib import HCL
>>> cols = HCL(H = [160, 210, 260, 310, 360],
>>>    C = [ 70,  40,  10,  40,  70],
>>>    L = [ 50,  70,  90,  70,  50])
>>> cols.swatchplot()
```

class colorlib.**hexcols**(*hex_*)
Color object for hex colors.

Takes up a set of hex colors. Can be converted to all other color spaces including *RGB*, *sRGB*, *HLS*, and *"hex"* (*hexcols*), *CIEXYZ*, *CIELUV*, *CIELAB*, *polarLUV*, *polarLAB*,

Parameters **hex**(*str, list of str, or numpy.ndarray of type str*) – hex colors. Only six and eight digit hex colors are allowed (e.g., `#000000` or `#00000050` if with alpha value). If invalid hex colors are provided the object will raise a ValueError. Input can be a single string, a list of strings, or a *numpy.ndarray* containing a set of hex colors. Invalid hex colors will be handled as *numpy.nan*, alpha values can be provided but will be ignored

Examples

```
>>> from colorspace.colorlib import hexcols
>>> c = hexcols("#cecece")
>>> c = hexcols(["#ff0000", "#00ff00"])
>>> from numpy import asarray
>>> c = hexcols(asarray(["#ff000030", "#00ff0030"], dtype = "|S9"))
>>> #Convert hex colors to another color space (CIEXYZ for example):
>>> c.to("CIEXYZ")
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)
Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary

class `colorlib.`**sRGB**(*R*, *G*, *B*, *alpha = None*, *gamma = None*)

sRGB (device dependent RGB) color object.

Allowes conversions to: *CIEXYZ*, *CIELUV*, *CIELAB*, *RGB*, *HSV*, *HLS*, *polarLUV*, *polarLAB*, *"hex"* (*hexcols*). Allows additional alpha input. Note that the alpha channel will not be modified but preserved. R, G, and B have to be within `[0., 1.]`.

Parameters

- **R** (*float, list of floats, numpy.ndarray*) – intensity of red `[0.,1.]`
- **G** (*float, list of floats, numpy.ndarray*) – intensity of green `[0.,1]`
- **B** (*float, list of floats, numpy.ndarray*) – intensity of blue `[0.,1]`
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added
- **gamma** (*None, float*) – gamma parameter. Used to convert from device dependent sRGB to RGB. If not set the default of 2.4 is used

Examples

```
>>> from colorspace.colorlib import sRGB
>>> c = sRGB(1., 0.3, 0.5)
>>> c = sRGB([1.,0.8], [0.5,0.5], [0.0,0.2])
>>>
>>> from numpy import asarray
>>> c = sRGB(asarray([1.,0.8]), asarray([0.5,0.5]), asarray([0.0,0.2]))
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)

Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary.

class colorlib.**RGB**(*R*, *G*, *B*, *alpha = None*)

Device independent RGB color object.

Allowes conversions to: *CIEXYZ*, *CIELUV*, *CIELAB*, *sRGB*, *HSV*, *HLS*, *polarLUV*, *polarLAB*, *"hex"* (*hexcols*). Allows additional alpha input. Note that the alpha channel will not be modified but preserved.

Parameters

- **R** (*numeric*) – intensity of red `[0.,1.]`
- **G** (*numeric*) – intensity of green `[0.,1.]`
- **B** (*numeric*) – intensity of blue `[0.,1.]`
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added

Examples

```
>>> from colorspace.colorlib import sRGB
>>> c = sRGB(1., 0.3, 0.5)
>>> c = sRGB([1.,0.8], [0.5,0.5], [0.0,0.2])
>>>
>>> from numpy import asarray
>>> c = sRGB(asarray([1.,0.8]), asarray([0.5,0.5]), asarray([0.0,0.2]))
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)

Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary.

class colorlib.**polarLUV**(*H*, *C*, *L*, *alpha = None*)

polarLUV or HCL color object. The polar representation of the CIELUV (`colorspace.CIELUV`) color space is also known as Hue-Chroma-Luminance (HCL) color space. polarLUV colors can be converted into: CIEXYZ , CIELUV , CIELAB , RGB , sRGB , polarLAB , hex. Not allowed (ambiguous) to convert into: HSV, HLS.

Parameters

- **L** (*numeric*) – single value or vector for hue dimension `[-360.,360.]`
- **U** (*numeric*) – single value or vector for chroma dimension `[0., 100.+]`
- **V** (*numeric*) – single value or vector for luminance dimension `[0., 100.]`
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added

Examples

```
>>> from colorspace.colorlib import polarLUV, HCL
>>> c = polarLUV(100., 30, 50.)
>>> c = HCL(100., 30, 50.) # Equivalent to the command above
>>> c = HCL([100.], [30.], [50.])
>>> c = HCL([100, 80], [30,50], [30,80])
>>> from numpy import asarray
>>> c = HCL(asarray([100,80]), asarray([30,50]), asarray([30,80]))
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)

Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., CIEXYZ, HCL, hex, RGB, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary

colorlib.**HCL**

alias of *colorlib.polarLUV*

class colorlib.**CIELUV**(*L*, *U*, *V*, *alpha = None*)

CIELUV color object.

polarLUV colors can be converted into: CIEXYZ, CIELUV, CIELAB, RGB, sRGB, polarLAB, hex. Not allowed (ambiguous) to convert into: HSV, HLS.

Parameters

- **L** (*numeric*) – single value or multiple values for L-dimension
- **U** (*numeric*) – single value or multiple values for U-dimension
- **V** (*numeric*) – single value or multiple values for V-dimension
- **alpha** (*None or numeric*) – single value or vector of numerics in [0.,1.] for the alpha channel (0. means transparent, 1. opaque). If None no transparency is added

Examples

```
>>> from colorspace.colorlib import CIELUV
>>> c = CIELUV(0, 10, 10)
>>> c = CIELUV([10, 30], [20, 80], [100, 40])
>>> from numpy import asarray
>>> c = CIELUV(asarray([10, 30]), asarray([20, 80]), asarray([100, 40]))
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to (*to*, *fixup = True*)
Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary

class colorlib.**CIEXYZ** (*X*, *Y*, *Z*, *alpha = None*)
CIEXYZ color object.

Allowes conversions to: *CIEXYZ*, *CIELUV*, *CIELAB*, *RGB*, *polarLUV*, *polarLAB*, *hexcols*. Not possible are conversions to (ambiguous): *HSV*, *HLS*.

Parameters

- **X** (*numeric*) – single value or multiple values for X-dimension
- **Y** (*numeric*) – single value or multiple values for Y-dimension
- **Z** (*numeric*) – single value or multiple values for Z-dimension
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added

Examples

```
>>> from colorspace.colorlib import CIEXYZ
>>> c = CIEXYZ(80, 30, 10)
>>> c = CIEXYZ([10, 0], [20, 80], [40, 40])
>>> from numpy import asarray
>>> c = CIEXYZ(asarray([10, 0]), asarray([20, 80]), asarray([40, 40]))
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to (*to*, *fixup = True*)
Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary

class colorlib.**CIELAB** (*L*, *A*, *B*, *alpha = None*)
CIELAB color object.

Allowes conversions to: *CIEXYZ*, *CIELUV*, *CIELAB*, *RGB*, *polarLUV*, *polarLAB*, *"hex"* (*hexcols*). Not possible are conversions to (ambiguous): *HSV*, *HLS*.

Parameters

- **L** (*numeric*) – single value or multiple values for L dimension
- **A** (*numeric*) – single value or multiple values for A dimension
- **B** (*numeric*) – single value or multiple values for B dimension
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added

Examples

```
>>> from colorspace.colorlib import CIELAB
>>> c = CIELAB(-30, 10, 10)
>>> c = CIELAB([-30, 30], [20, 80], [40, 40])
>>> from numpy import asarray
>>> c = CIELAB(asarray([-30, 30]), asarray([20, 80]), asarray([40, 40]))
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)

Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary

class colorlib.**polarLAB**(*L*, *A*, *B*, *alpha = None*)

polarLAB color object.

Allowes conversions to: *CIEXYZ*, *CIELUV*, *CIELAB*, *RGB*, *polarLUV*, *polarLAB*, *"hex"* (*hexcols*). Not possible are conversions to (ambiguous): *HSV*, *HLS*.

Parameters

- **L** (*numeric*) – single value or multiple values for L dimension
- **A** (*numeric*) – single value or multiple values for A dimension
- **B** (*numeric*) – single value or multiple values for B dimension
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added

:param .. seealso::: This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)

Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary.

class colorlib.**HLS**(*H*, *L*, *S*, *alpha = None*)

HLS (Hue-Lightness-Saturation) color space.

Allowes conversions to: *RGB*, *sRGB*, *HLS*, and *"hex"* (*hexcols*). Not possible are conversions to (ambiguous): *CIEXYZ*, *CIELUV*, *CIELAB*, *polarLUV*, *polarLAB*,

Parameters

- **H** (*numeric*) – single value or multiple values for the hue dimension
- **L** (*numeric*) – single value or multiple values for the lightness dimension
- **S** (*numeric*) – single value or multiple values for the saturation dimension
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added

Examples

```
>>> from colorspace.colorlib import HLS
>>> cols = HLS([150, 0, 10], [0.1, 0.7, 0.1], [3, 0, 3])
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)

Transforms the colors into a new color space, if possible.

No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary.

class colorlib.**HSV**(*H*, *S*, *V*, *alpha = None*)

HSV (Hue-Saturation-Value) color object.

Allowes conversions to: *RGB*, *sRGB*, *HLS*, and *"hex"* (*hexcols*). Not possible are conversions to (ambiguous): *CIEXYZ*, *CIELUV*, *CIELAB*, *polarLUV*, *polarLAB*,

Parameters

- **H** (*numeric*) – single value or multiple values for the hue dimension
- **S** (*numeric*) – single value or multiple values for the saturation dimension

- **V** (*numeric*) – single value or multiple values for the value dimension
- **alpha** (*None or numeric*) – single value or vector of numerics in `[0.,1.]` for the alpha channel (`0.` means transparent, `1.` opaque). If `None` no transparency is added

Examples

```
>>> from colorspace.colorlib import HSV
>>> cols = HSV([150, 150, 10], [1.5, 0, 1.5], [0.1, 0.7, 0.1])
```

See also:

This object extens the *colorlib.colorobject* which provides some methods to e.g., extract color or to modify the whitepoint.

to(*to*, *fixup = True*)

Transforms the colors into a new color space, if possible.

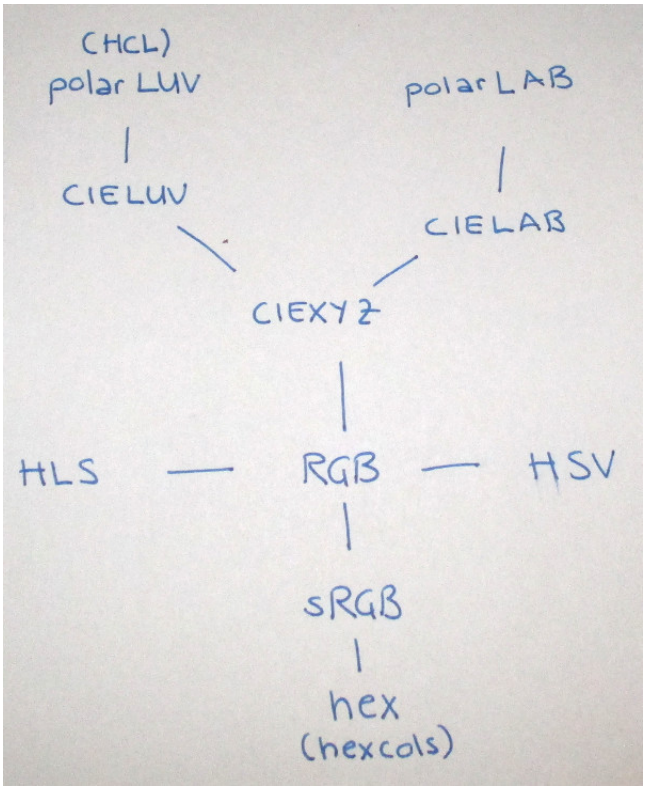
No return, converts the object into a new color space and modifies the underlying object. After calling this method the original object will be of a different type.

Parameters

- **to** (*str*) – name of the color space into which the colors should be converted (e.g., `CIEXYZ`, `HCL`, `hex`, `RGB`, ...)
- **fixup** (*bool*) – whether or not colors outside the defined rgb color space should be corrected if necessary.

1.4.4 The colorlib

The *colorlib* module is handling the transformation between different color spaces. A set of different color spaces is available including *colorlib.CIELUV*, *colorlib.CIELAB*, *colorlib.CIEXYZ*, *colorlib.polarLUV*/*colorlib.HCL*, *colorlib.polarLAB*, *colorlib.HSV*, *colorlib.HLS*, *colorlib.RGB*, *colorlib.sRGB*, and *colorlib.hexcols* for HEX colors.



class colorlib.**colorlib**

The colorlib class is a collection of methods used to convert or transform colors between different color spaces.

DEG2RAD(*x*)

ParameterConver degrees into radiant.

Parameters **x** (*float or array of floats*) – values in degrees

Returns Returns input x in radiant.

Return type float or float array

DEVRGB_to_RGB(*R*, *G*, *B*, *gamma = 2.4*)

Device dependent sRGB to device independent RGB.

Parameters

- **R** (*numpy.ndarray*) – indensities for red ([0.,1.])
- **G** (*numpy.ndarray*) – indensities for green ([0.,1.])
- **B** (*numpy.ndarray*) – indensities for blue ([0.,1.])
- **gamma** (*float*) – gamma adjustment.

Returns Returns a list of *numpy.ndarrays* with adjusted R, G, and B values.

Return type list

HLAB_to_XYZ(*L*, *A*, *B*, *XN = None*, *YN = None*, *ZN = None*)

Hunter LAB to CIE-XYZ.

Note: Note that the Hunter LAB is no longer part of the public API, but the code is still here in case needed.

Parameters

- **L** (*numpy.ndarray*) – values for the L dimension
- **A** (*numpy.ndarray*) – values for the A dimension
- **B** (*numpy.ndarray*) – values for the B dimension
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three NA) default values will be used

Returns Returns corresponding CIE-XYZ chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[X, Y, Z]`).

Return type list

HLS_to_RGB(*h*, *l*, *s*)
Convert RLS to HLS.

All r/g/b values in `[0.,1.]`, h in `[[0., 360.]`, l and s in `[0., 1.]`. From: http://wiki.beyondunreal.com/wiki/RGB_To_HLS_Conversion.

Parameters

- **h** (*nympy.ndarray*) – hue values
- **l** (*numpy.ndarray*) – lightness
- **s** (*numpy.ndarray*) – saturation

Returns Returns a *numpy.ndarray* with the corresponding coordinates in the RGB color space (`[r, g, b]`). Same length as the inputs.

Return type list

HSV_to_RGB(*r*, *g*, *b*)
Convert RGB to HSV.

Parameters

- **h** (*nympy.ndarray*) – hue values
- **s** (*numpy.ndarray*) – saturation
- **v** (*numpy.ndarray*) – value (the value-dimension of HSV)

Returns Returns a *numpy.ndarray* with the corresponding coordinates in the RGB color space (`[r, g, b]`). Same length as the inputs.

Return type list

LAB_to_XYZ(*L*, *A*, *B*, *XN* = *None*, *YN* = *None*, *ZN* = *None*)
CIELAB to CIEXYZ.

Parameters

- **L** (*numpy.ndarray*) – values for the L dimension
- **A** (*numpy.ndarray*) – values for the A dimension
- **B** (*numpy.ndarray*) – values for the B dimension
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three NA) default values will be used

Returns Returns corresponding X/Y/Z coordinates of CIE chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[X, Y, Z]`).

Return type list

LAB_to_polarLAB(*L*, *A*, *B*)
Convert from CIELAB to the polar representation polarLAB.

Parameters

- **L** (*numpy.ndarray*) – values for the L dimension of the CIELAB color space
- **A** (*numpy.ndarray*) – values for the A dimension of the CIELAB color space
- **B** (*numpy.ndarray*) – values for the B dimension of the CIELAB color space

Returns Returns corresponding polar LAB chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[L, A, B]`).

Return type list

LUV_to_XYZ(*L*, *U*, *V*, *XN* = *None*, *YN* = *None*, *ZN* = *None*)
CIE-LUV to CIE-XYZ.

Parameters

- **L** (*numpy.ndarray*) – values for the L dimension
- **U** (*numpy.ndarray*) – values for the U dimension
- **V** (*numpy.ndarray*) – values for the V dimension
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three NA) default values will be used

Returns Returns a list of CIE-XYZ coordinates (`[X, Y, Z]`) with the same length as the input arrays.

Return type list

LUV_to_polarLUV(*L*, *U*, *V*)
LUV to polarLUV (HCL).

Parameters

- **L** (*numpy.ndarray*) – values for the X dimension
- **U** (*numpy.ndarray*) – values for the Y dimension
- **V** (*numpy.ndarray*) – values for the Z dimension

Returns Returns a list of polarLUV or HCL coordinates (`[L, C, H]`) with the same length as the input arrays. The HCL color space is simply the polar representation of the CIE-LUV color space.

Return type list

RAD2DEG(*x*)
ParameterConver radiant to degrees.

Parameters **x** (*float or array of floats*) – values in radiant

Returns Returns input `x` in degrees.

Return type float or array of floats

RGB_to_DEVRGB(*R*, *G*, *B*, *gamma=2.4*)
DEVRGB_to_RGB(R, G, B, gamma = 2.4)

Device independent RGB to device dependent sRGB.

Parameters

- **R** (*numpy.ndarray*) – indensities for red (`[0.,1.]`).
- **G** (*numpy.ndarray*) – indensities for green (`[0.,1.]`).
- **B** (*numpy.ndarray*) – indensities for blue (`[0.,1.]`).
- **gamma** (*float*) – gamma adjustment.

Returns Returns a list of *numpy.ndarrays* with adjusted R, G, and B values.

Return type list

RGB_to_HLS(*r*, *g*, *b*)
Convert RGB to HLS.

All r/g/b values in `[0.,1.]`, h in `[[0., 360.]`, l and s in `[0., 1.]`. From: http://wiki.beyondunreal.com/wiki/RGB_To_HLS_Conversion.

Parameters

- **r** (*numpy.ndarray*) – intensities for red (`[0.,1.]`)
- **g** (*numpy.ndarray*) – intensities for green (`[0.,1.]`)
- **b** (*numpy.ndarray*) – intensities for blue (`[0.,1.]`)

Returns Returns a *numpy.ndarray* with the corresponding coordinates in the HLS color space (`[h, l, s]`). Same length as the inputs.

Return type list

RGB_to_HSV(*r*, *g*, *b*)
Convert RGB to HSV.

Parameters

- **r** (*numpy.ndarray*) – intensities for red (`[0.,1.]`)
- **g** (*numpy.ndarray*) – intensities for green (`[0.,1.]`)
- **b** (*numpy.ndarray*) – intensities for blue (`[0.,1.]`)

Returns Returns a *numpy.ndarray* with the corresponding coordinates in the HSV color space (`[h, s, v]`). Same length as the inputs.

Return type list

RGB_to_XYZ(*R*, *G*, *B*, *XN = None*, *YN = None*, *ZN = None*)
Device independent RGB to XYZ.

R, G, and B give the levels of red, green and blue as values in the interval `[0.,1.]`. X, Y and Z give the CIE chromaticies.

Parameters

- **R** (*numpy.ndarray*) – indensities for red (`[0.,1.]`)
- **G** (*numpy.ndarray*) – indensities for green (`[0.,1.]`)
- **B** (*numpy.ndarray*) – indensities for blue (`[0.,1.]`)

- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three `None`) default values will be used

Returns Returns corresponding X/Y/Z coordinates of CIE chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[X, Y, Z]`).

Return type list

XYZ_to_HLAB(*X*, *Y*, *Z*, *XN* = *None*, *YN* = *None*, *ZN* = *None*)
CIE-XYZ to Hunter LAB.

Note: Note that the Hunter LAB is no longer part of the public API, but the code is still here in case needed.

Parameters

- **X** (*numpy.ndarray*) – values for the Z dimension
- **Y** (*numpy.ndarray*) – values for the Z dimension
- **Z** (*numpy.ndarray*) – values for the Z dimension
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three `None`) default values will be used

Returns Returns corresponding Hunter LAB chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[L, A, B]`).

Return type list

XYZ_to_LAB(*X*, *Y*, *Z*, *XN* = *None*, *YN* = *None*, *ZN* = *None*)
CIEXYZ to CIELAB.

Parameters

- **X** (*numpy.ndarray*) – values for the X dimension
- **Y** (*numpy.ndarray*) – values for the Y dimension
- **Z** (*numpy.ndarray*) – values for the Z dimension
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three NA) default values will be used

Returns Returns corresponding L/A/B coordinates, a list of *numpy.ndarray*'s of the same length as the inputs (`[L, A, B]`).

Return type list

XYZ_to_LUV(*X*, *Y*, *Z*, *XN* = *None*, *YN* = *None*, *ZN* = *None*)
CIE-XYZ to CIE-LUV.

Parameters

- **X** (*numpy.ndarray*) – values for the X dimension
- **Y** (*numpy.ndarray*) – values for the Y dimension
- **Z** (*numpy.ndarray*) – values for the Z dimension

- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three NA) default values will be used.

Returns Returns a list of CIELUV coordinates (`[L, U, V]`) with the same length as the input arrays.

Return type list

XYZ_to_RGB(*X*, *Y*, *Z*, *XN = None*, *YN = None*, *ZN = None*)
CIEXYZ to device independent RGB.

R, G, and B give the levels of red, green and blue as values in the interval `[0.,1.]`. X, Y and Z give the CIE chromaticies.

Parameters

- **X** (*numpy.ndarray*) – values for the Z dimension
- **Y** (*numpy.ndarray*) – values for the Z dimension
- **Z** (*numpy.ndarray*) – values for the Z dimension
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three `None`) default values will be used

Returns Returns corresponding X/Y/Z coordinates of CIE chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[R, G, B]`).

Return type list

XYZ_to_sRGB(*X*, *Y*, *Z*, *XN = None*, *YN = None*, *ZN = None*)
CIEXYZ to sRGB.

R, G, and B give the levels of red, green and blue as values in the interval `[0.,1.]`. X, Y and Z give the CIE chromaticies.

Parameters

- **X** (*numpy.ndarray*) – values for the Z dimension
- **Y** (*numpy.ndarray*) – values for the Z dimension
- **Z** (*numpy.ndarray*) – values for the Z dimension
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three NA) default values will be used

Returns Returns corresponding X/Y/Z coordinates of CIE chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[R, G, B]`).

Return type list

XYZ_to_uv(*X*, *Y*, *Z*)
CIE-XYZ to u and v.

Parameters

- **X** (*numpy.ndarray*) – values for the Z dimension.
- **Y** (*numpy.ndarray*) – values for the Y dimension.
- **Z** (*numpy.ndarray*) – values for the Z dimension.

Returns Returns a list of *numpy.ndarrays* containing u and v (`[u, v]`).

Return type list

ftrans(*u*, *gamma*)
Gamma Correction.

Function `gtrans()` and `ftrans()` provide gamma correction between the RGB (device independent) and sRGB (device dependent) color space.

The standard value of gamma for sRGB displays is approximately `2.2`, but more accurately is a combination of a linear transform and a power transform with exponent `2.4`. `gtrans()` maps linearized sRGB to sRGB, `ftrans()` provides the inverse map.

Parameters

- **u** (*numpy.ndarray*) – float array of length N
- **gamma** (*float or numpy.ndarray*) – gamma value. If float or *numpy.ndarray* of length one gamma will be recycled (if length `u > 1`)

Returns Same length as input `u`.

Return type numpy.ndarray

gtrans(*u*, *gamma*)
Gamma Correction.

Function `gtrans` and `ftrans` provide gamma correction which can be used to switch between sRGB and linearized sRGB (RGB).

The standard value of gamma for sRGB displays is approximately `2.2`, but more accurately is a combination of a linear transform and a power transform with exponent `2.4` gtrans maps linearized sRGB to sRGB, ftrans provides the inverse map.

Parameters

- **u** (*numpy.ndarray*) – float array of length N
- **gamma** (*float or numpy.ndarray*) – gamma value. If float or *numpy.ndarray* of length one gamma will be recycled (if length `u > 1`)

Returns Same length as input `u`.

Return type numpy.ndarray

hex_to_sRGB(*hex_*, *gamma = 2.4*)
Convert hex colors to sRGB.

Parameters

- **hex** (*str, list of str*) – hex strings.
- **gamma** (*float*) – gamma correction factor.

Returns Returns a list of numpy.ndarrays with the corresponding red, green, and blue intensities (`[0.,1.]`).

Return type list

polarLAB_to_LAB(*L*, *A*, *B*)
Convert form polarLAB to onvert CIELAB.

Parameters

- **L** (*numpy.ndarray*) – values for the L dimension of the polar LAB color space
- **A** (*numpy.ndarray*) – values for the A dimension of the polar LAB color space

- **B** (*numpy.ndarray*) – values for the B dimension of the polar LAB color space

Returns Returns corresponding CIELAB chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[L, A, B]`).

Return type list

polarLUV_to_LUV(*L*, *C*, *H*)
polarLUV (HCL) to LUV.

Parameters

- **L** (*numpy.ndarray*) – values for the L or luminance dimension
- **C** (*numpy.ndarray*) – values for the C or chroma dimension
- **H** (*numpy.ndarray*) – values for the H or hue dimension

Returns Returns a list of polarLUV or HCL coordinates (`[L, U, V]`) with the same length as the input arrays.

Return type list

sRGB_to_XYZ(*R*, *G*, *B*, *XN = None*, *YN = None*, *ZN = None*)
sRGB to CIEXYZ.

R, G, and B give the levels of red, green and blue as values in the interval `[0.,1.]`. X, Y and Z give the CIE chromaticies.

Parameters

- **R** (*numpy.ndarray*) – indensities for red (`[0.,1.]`)
- **G** (*numpy.ndarray*) – indensities for green (`[0.,1.]`)
- **B** (*numpy.ndarray*) – indensities for blue (`[0.,1.]`)
- **YN, ZN** (*XN,*) – chromaticity of the white point. If of length 1 the white point specification will be recycled if length of R/G/B is larger than one. If not specified (all three `None`) default values will be used

Returns Returns corresponding X/Y/Z coordinates of CIE chromaticies, a list of *numpy.ndarray*'s of the same length as the inputs (`[X, Y, Z]`).

Return type list

sRGB_to_hex(*r*, *g*, *fixup = True*)
sRGB colors to hex colors.

Parameters

- **r** (*numpy.ndarray*) – intensities for red (`[0.,1.,]`)
- **g** (*numpy.ndarray*) – intensities for green (`[0.,1.,]`)
- **b** (*numpy.ndarray*) – intensities for blue (`[0.,1.,]`)
- **fixup** (*bool*) – whether or not the rgb values should be corrected if they lie outside the defined RGB space (outside `[0.,1.,]`)

Returns A list with hex colors as strings.

Return type list

1.4.5 Spectrum Plot

specplot.**specplot**(*hex_*, *rgb = True*, *hcl = True*, *palette = True*, *fix = True*, ***kwargs*)

Visualization of the RGB and HCL spectrum given a set of hex colors. As the hues for low-chroma colors are not (or poorly) identified, by default a smoothing is applied to the hues (`fix = TRUE`). Also, to avoid jumps from 0 to 360 or vice versa, the hue coordinates are shifted suitably.

No return, creates an interactive figure.

Parameters

- **hex** (*list or numpy.ndarray*) – hex color codes.
- **hcl** (*bool*) – whether or not to plot the HCL color spectrum.
- **palette** (*bool*) – whether or not to plot the colors as a color map.
- **fix** (*bool*) – should the hues be fixed to be on a smooth(er) curve? Details in the method description.
- **rgb** (*bool*) – whether or not to plot the RGB color spectrum. Default is False.
- **kwargs** – Currently not used.

Example

```
>>> from colorspace import rainbow_hcl
>>> from colorspace import specplot
>>> pal = rainbow_hcl(100)
>>> specplot(pal.colors())
>>> specplot(pal.colors(), rgb = False, hcl = True, palette = False)
```

Todo: Implement the smoothings to improve the look of the plots. Only partially implemented, the spline smoother is missing.

1.4.6 Choose Palette

class choose_palette.**Slider**(*x*, *y*, *width*, *height*, *active*, *type_*, *from_*, *to*, *resolution*, ***kwargs*)

Initializes a new Slider object for the graphical user interface *choose_palette.gui*. A Slider is a combination of a `Tk.Frame` including a `Tk.Label`, `Tk.Slider`, and a `Tk.Entry` element with all necessary interactions.

Parameters

- **x** (*int*) – x-position on the Tk interface
- **y** (*int*) – y-position on the Tk interface
- **width** (*int*) – width of the Slider object (`Tk.Frame` taking up `Tk.Scale`, `Tk.Label`, and `Tk.Entry`)
- **height** (*int*) – height of the Slider object (`Tk.Frame` taking up `Tk.Scale`, `Tk.Label`, and `Tk.Entry`)
- **type** (*str*) – name of the Slider
- **from** (*numeric*) – lower value of the Slider (see *isValidInt()*, *isValidFloat()*)

- **to** (*numeric*) – upper value of the Slider (see *isValidInt()*, *isValidFloat()*)
- **resolution** (*numeric*) – resolution of the slider, the increments when moving the Slider
- **kwargs** – Unused

OnTrace(**args*, ***kwargs*)
Triggered when *Slider.trace()* is triggered. The method is loading the current value and sets the `Tk.Scale` and `Tk.Entry` element to the new value.

disable()
Disables the *Slider*.

enable()
Enables the *Slider*.

get()

Returns Returns the current value of the slider. The return value depends on the slider config (*int* or *float*).

Return type int or float

isValidFloat(*x*, *from_ = -999.*, *to = 999.*)
Helper function to check whether `x` is a valid float in the range `[from_,to]`.

Parameters

- **x** (*float*) – Value to be validated
- **from** (*float*) – Lower limit of the valid range
- **to** (*float*) – Upper limit of the valid range

Returns Returns `True` if `x` is a valid float within `[from_, to]` and `False` otherwise.

Return type bool

isValidInt(*x*, *from_ = -999*, *to = 999*)
Helper function to check whether `x` is a valid integer in the range `[from_,to]`.

Parameters

- **x** (*int*) – Value to be validated
- **from** (*int*) – Lower limit of the valid range
- **to** (*int*) – Upper limit of the valid range

Returns Returns `True` if `x` is a valid float within `[from_, to]` and `False` otherwise.

Return type bool

is_active()
Returns `True` if the *Slider* is active and `False` if currently inactive.

name()

Returns Returns the name of the *Slider*.

Return type str

trace(*mode*, **args*, ***kwargs*)
Trace method of the *Slider* object.

Parameters

- **mode** (*str*) – default is `w` (call observer when variable is written)
- **args** – arguments passed to `Tkinter.<vartype>.trace()`, at least one argument (a callback function) should be provided
- **kwargs** – arguments passed to `Tkinter.<vartype>.trace()`, unused

choose_palette.**choose_palette**(***kwargs*)

Graphical user interface to choose HCL based color palettes. Returns an object of *palettes.diverging_hcl*, *palettes.qualitative_hcl*, or *palettes.sequential_hcl* with user-defined default settings.

Parameters **kwargs** – See *choose_palette.gui*.

Returns The object allows to get colors in different ways, the default is a list with hex colors. See *palettes.hclpalette* or, more specifically, the manual of the depending palette (*palettes.diverging_hcl*, *palettes.qualitative_hcl*, or *palettes.sequential_hcl*).

Return type *palettes.hclpalette* object

class choose_palette.**currentpalettecanvas**(*parent*, *x*, *y*, *width*, *height*)

Draws the current palette (the palette as specified on the GUI), will be displayed in the lower part of the GUI.

Parameters

- **parent** (`Tk`) – the `Tk` object (interface)
- **x** (*numeric*) – x position on the interface
- **y** (*numeric*) – y position on the interface
- **width** (*numeric*) – width of the palette on the interface
- **height** (*numeric*) – height of the palette on the interface

class choose_palette.**defaultpalettecanvas**(*palframe*, *sliders*, *pal*, *n*, *xpos*, *figwidth*, *figheight*)

Sets up a `Tk.Canvas` element containing the colors of the default HCL color palettes which will be placed in the top part of the GUI.

Parameters

- **palframe** (`Tk.Frame`) – the bounding `Tk.Frame` which takes up the palettes.
- **sliders** (*list*) – list of *Slider* objects. When a user selets a new default palette the sliders will be set to the specification given the selected palette (and enabled/disabled corresponding to the palette specification)
- **pal** (`defaultpalette`) – the default color palette
- **n** (*int*) – number of colors to be drawn
- **xpos** (*numeric*) – x position within `Tk.Canvas` (palframe input)
- **figwidth** (*numeric*) – width of the `Tk.Canvas` element (palframe input)
- **figheight** (*numeric*) – width of the `Tk.Canvas` element (palframe input)

class choose_palette.**gui**(***kwargs*)

choose_palette(**kwargs)

Graphical user interface to choose custom HCL-basec color palettes.

Parameters **kwargs** – Optional, can be used to change the defaults when starting the GUI. Currently a parameter called `palette` is allowed to specify the initial color palette. If not set, `palette = "Blue-Red"` is used.

Example

```
>>> colorspace.choose_palette()
```

OnChange(**args*, ***kwargs*)
Triggered any time the slider values or control arguments change. Draws new current palette (see `_draw_currentpalette()`).

OnPaltypeChange(**args*, ***kwargs*)
The callback function of the drop down element. Triggered every time the drop down element changes.

control()

Returns Returns a dictionary with the current control options (see `_add_control()`).

Return type dict

get_colors()

Returns Returns a list of hex colors and `nan` given the current settings on the GUI. `numpy.nan` will be returned if `fixup` is set to False but some colors lie outside the RGB color space.

Return type list

master()

Returns Returns the `Tk` GUI object.

Return type `Tk`

method()

Returns Returns the name of the object which has to be called to get the colors. The name of the object is defined in the palconfig config files. For "Diverging" palettes this will be *palettes.diverging_hcl*, for "Qualitative" *palettes.qualitative_hcl*, and for "Sequential" palettes *palettes.sequential_hcl*.

Return type str

palettes()

Returns Returns the default palettes available.

Return type `palettes.hclpalettes`

palframe()

Returns Returns the palette frame (`Tk.Frame` object, see `_add_palframe()`).

Return type `Tk.Frame`

settings(**args*, ***kwargs*)
Used to load/store current palette settings (gui settings).

Parameters

- **args** – strings to load one/several parameters.
- **kwargs** – named arguments, used to store values.

Returns Returns a dictionary with the current slider settings.

Return type dict

sliders()

Returns List of *Slider* objects.

Return type list

1.4.7 Color Vision Deficiency

class CVD.**CVD**(*cols*, *type_*, *severity = 1.*)

Object to simulate color vision deficiencies (CVD) for protanope, deteranope, and tritanope visual constraints. There are wrapper functions to provide simple access for the users, see *deutan()*, *protan()*, and *tritan()*.

No return values, initializes a new CVD object which provides functions to manipulate the colors acording to the color deficiency (type_).

Parameters

- **cols** (list of str or colorobject) – a colorobject (such as RGB, HCL, CIEXYZ) or a list of hex colors
- **type** (*str*) – type of the deficiency which should be simulated. Currently allowed are deutan, protan, and tritan
- **severity** (*float*) – severity in [0., 1.]. Zero means no deficiency, one maximum deficiency

Examples

```
>>> from colorspace import rainbow_hcl
>>> cols = rainbow_hcl()(10)
>>> from colorspace.CVD import CVD
>>> deut = CVD(cols, "deutan")
>>> prot = CVD(cols, "protan")
>>> trit = CVD(cols, "tritan")
```

```
>>> from colorspace import specplot
>>> specplot(deut.colors())
>>> specplot(prot.colors())
>>> specplot(trit.colors())
```

colors()

Returns Returns the colors of the object with simulated colors for the color vision deficiency as specified when initializing the object.

Return type *colorobject*

deutan_cvd_matrizes(*s*)

Returns the transformation matrix to simpulate deuteranope color vision deficiency.

Parameters **s** (*int*) – an integer in [0, 11] to specify which matrix sould be returned

Returns Returns a numpy float matrix of shape 3 x 3. The color deficiency transformation or rotation matrix.

Return type numpy.ndarray

protan_cvd_matrizes(*s*)

Returns the transformation matrix to simpulate protanope color vision deficiency.

Parameters **s** (*int*) – an integer in `[0, 11]` to specify which matrix sould be returned

Returns Returns a numpy float matrix of shape `3 x 3`. The color deficiency transformation or rotation matrix.

Return type numpy.ndarray

tritan_cvd_matrizes(*s*)

Returns the transformation matrix to simpulate tritanope color vision deficiency.

Parameters **s** (*int*) – an integer in `[0, 11]` to specify which matrix sould be returned

Returns Returns a numpy float matrix of shape `3 x 3`. The color deficiency transformation or rotation matrix.

Return type numpy.ndarray

CVD.**desaturate**(*col*, *amount = 1.*)

Transform a vector of given colors to the corresponding colors with chroma reduced (by a tunable amount) in HCL space.

The colors of the color object `col` are transformed to the HCL color space. In HCL, In HCL, chroma is reduced and then the color is transformed back to a colorobject of the same class as the input.

Parameters

- **col** (*colorobject*) – a colorspace color object such as RGB, hexcols, CIELUV, ...
- **amount** (*float*) –

  a value in `[0.,1.]` defining the degree of desaturation. `amount = 1.` removes all color, `amount = 0.` none

Returns Returns a list of modified hex colors.

Return type list

Examples

```
>>> from colorspace import diverging_hcl
>>> from colorspace.colorlib import hexcols
>>> cols = hexcols(diverging_hcl()(10))
>>> from colorspace import specplot
>>> specplot(desaturate(cols))
>>> specplot(desaturate(cols, 0.5))
```

Todo: Handling of alpha values. And, in addition, add support for hex colors. Currently a list of hex colors as input is not allowed (fix it).

CVD.**deutan**(*cols*, *severity = 1.*)

Transformation of R colors by simulating color vision deficiencies, based on a CVD transform matrix. This function is a interface to the CVD object and returns simulated colors for deuteranope vision (green-yellow-red weakness).

Parameters

- **cols** (*list of str or colorobject*) – a colorobject (such as RGB, HCL, CIEXYZ) or a list of hex colors

- **severity** (*float*) – severity in `[0., 1.]`. Zero means no deficiency, one maximum deficiency

Returns Returns an object of the same type as the input object `cols` with modified colors as people with deuteranomaly see these colors (simulated).

Return type *colorobject*

Examples

```
>>> from colorspace import rainbow_hcl, specplot
>>> cols = rainbow_hcl()(100)
>>> specplot(cols)
>>> specplot(deutan(cols))
>>> specplot(deutan(cols, 0.5))
```

CVD.**protan**(*cols*, *severity = 1.*)

Transformation of R colors by simulating color vision deficiencies, based on a CVD transform matrix. This function is a interface to the CVD object and returns simulated colors for protanope vision.

Parameters

- **cols** (list of str or `colorobject`) – a colorobject (such as RGB, HCL, CIEXYZ) or a list of hex colors
- **severity** (*float*) – severity in `[0., 1.]`. Zero means no deficiency, one maximum deficiency

Returns Returns an object of the same type as the input object `cols` with modified colors as people with protanope color vision might see the colors (simulated).

Return type *colorobject*

Examples

```
>>> from colorspace import rainbow_hcl, specplot
>>> cols = rainbow_hcl()(100)
>>> specplot(cols)
>>> specplot(protan(cols))
>>> specplot(protan(cols, 0.5))
```

CVD.**tritan**(*cols*, *severity = 1.*)

Transformation of R colors by simulating color vision deficiencies, based on a CVD transform matrix. This function is a interface to the CVD object and returns simulated colors for tritanope vision.

Parameters

- **cols** (list of str or `colorobject`) – a colorobject (such as RGB, HCL, CIEXYZ) or a list of hex colors
- **severity** (*float*) – severity in `[0., 1.]`. Zero means no deficiency, one maximum deficiency

Returns Returns an object of the same type as the input object `cols` with modified colors as people with tritanomaly see these colors (simulated).

Return type *colorobject*

Examples

```
>>> from colorspace import rainbow_hcl, specplot
>>> cols = rainbow_hcl()(100)
>>> specplot(cols)
>>> specplot(tritan(cols))
>>> specplot(tritan(cols, 0.5))
```

CHAPTER 2

Other Packages and Further Reading

More information and further reading:

- hclwizard.org: more information about the HCL color space, introduction to the colorspace packages (in R and python), and some interactive tools to define effective HCL-based color palettes, pick colors, and check existing plots and figures for possible problems in terms of color vision deficiencies.
- A list of scientific articles which provide more detailed insights, e.g.
- The end of the rainbow: an open letter to the climate science community by Ed Hawkins, Doug McNeall, David Stephenson, Jonny Williams & Dave Carlson.
- Better Figures: Constructive criticism of the graphics of climate science by Doug McNeall.

Scientific articles with more detailed insights:

- Stauffer, R., Mayr, G. J., Dabernig, M., & Zeileis, A. (2015). Somewhere Over the Rainbow: How to Make Effective Use o f Colors in Meteorological Visualizations. Bulletin of the American Meteorological Society, 96(2), 203–216, doi: 10.1175/BAMS-D-13-00155.1.
- Zeileis, A., Hornik, K., & Murrell, P. (2009). Escaping RGBland: Selecting colors for statistical graphics. Computational Statistics &Amp; Data Analysis , 53(9), 3259–3270, doi:10.1016/j.csda.2008.11.033.
- Ihaka, R., 2003. Colour for presentation graphics. In: Hornik, K., Leisch, F., Zeileis, A. (Eds.), Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria, ISSN 1609-395X, URL: http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/Ihaka.pdf.
- And others (hclwizard.org reference list).

Some other packages providing color maps in python (on top of the default color maps) wich might be of interest:

- seaborn: statistical data visualization. The package also provides access to a range of (mostly) well specified. color palettes.
- palettable: color palettes for python. Formely known as `brewer2mpl`. Provides a range of color palettes including “Brewer2” and “Carto” palettes.
- ColorBrewer2.org: the source of the brewer colors, interactive webpage by Cynthia Brewer, Mark Harrower and The Pennsylvania State University.

CHAPTER 3

Known issues

Warning: White point implemented but might require some additional testing.

CHAPTER 4

TODO's

Todo: Introduction to the three basic principles of the diverging, sequential, and qualitative color maps.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-colorspace/checkouts/latest/docs/hclcolorspace.rst, line 107.)

Todo: Handling of alpha values. And, in addition, add support for hex colors. Currently a list of hex colors as input is not allowed (fix it).

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-colorspace/checkouts/latest/colorspace/CVD.py:docstring of CVD.desaturate, line 27.)

Todo: Implement the smoothings to improve the look of the plots. Only partially implemented, the spline smoother is missing.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-colorspace/checkouts/latest/colorspace/specplot.py:docstring of specplot.specplot, line 31.)

Python Module Index

C

S

Index