python-colorspace Documentation

Release 0.0.1

Reto Stauffer

Sep 17, 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 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 mainly based on the technical specification of digital screens such as computer screens or TVs. The color of each individual pixel on a screen is created by mixing intensities of red, green, and blue (additive color mixture).

In contrast, the Hue-Chroma-Luminance color space is based on how human color perception works.

Path trough the HCL space

The animation shows the HCL color space as a volume. 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) shows defines the hue, the radial distance to the center the chroma.

The solid line inside the volume shows the path of the default `diverging_hcl` color map (with `n=11` colors) trough the HCL color space.

Importance of the luminance dimension

This simple example shows the effect of the luminance information. Even if additional color coding is used on top the luminance information is still processed by our brain and can either support the reader if used in an effective way, or make it hard or even impossible to gather the most important information if used without caution.

Effective Color Palettes

The colorspace package provides an easy to use interface to choose effective color maps based on the HCL color space.

Todo: To be done . . .

1.1.3 Usage Examples

Note: To be done . . .

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" `diverging_hcl` color palette and the "Purple-Orange" `sequential_hcl` color palette in the top row, and the "Set 2" `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 plut such a data set. The `qualitative_hcl` color map has iso-chroma (color intensity) and iso-luminance (lightness) and only varies in hue (the color itself). Such palettes are made for classification tasks and not to display a data set as shown in the demo plot. The `gist_ncar` palette should also not be used due to the immense discontinuity across the palette. More information about *effective color palettes* can be found *on this page*.

Color Vision Deficiency

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

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
    ----------
    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)

    fig.tight_layout()
```

(continues on next page)

(continued from previous page)

```
    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 handling methods are different to allow to use the color maps in python in an easy and generalized 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 tuch with me, helping hands would be very welcome)!

Version 0.0.1 (Sept 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 objets 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.

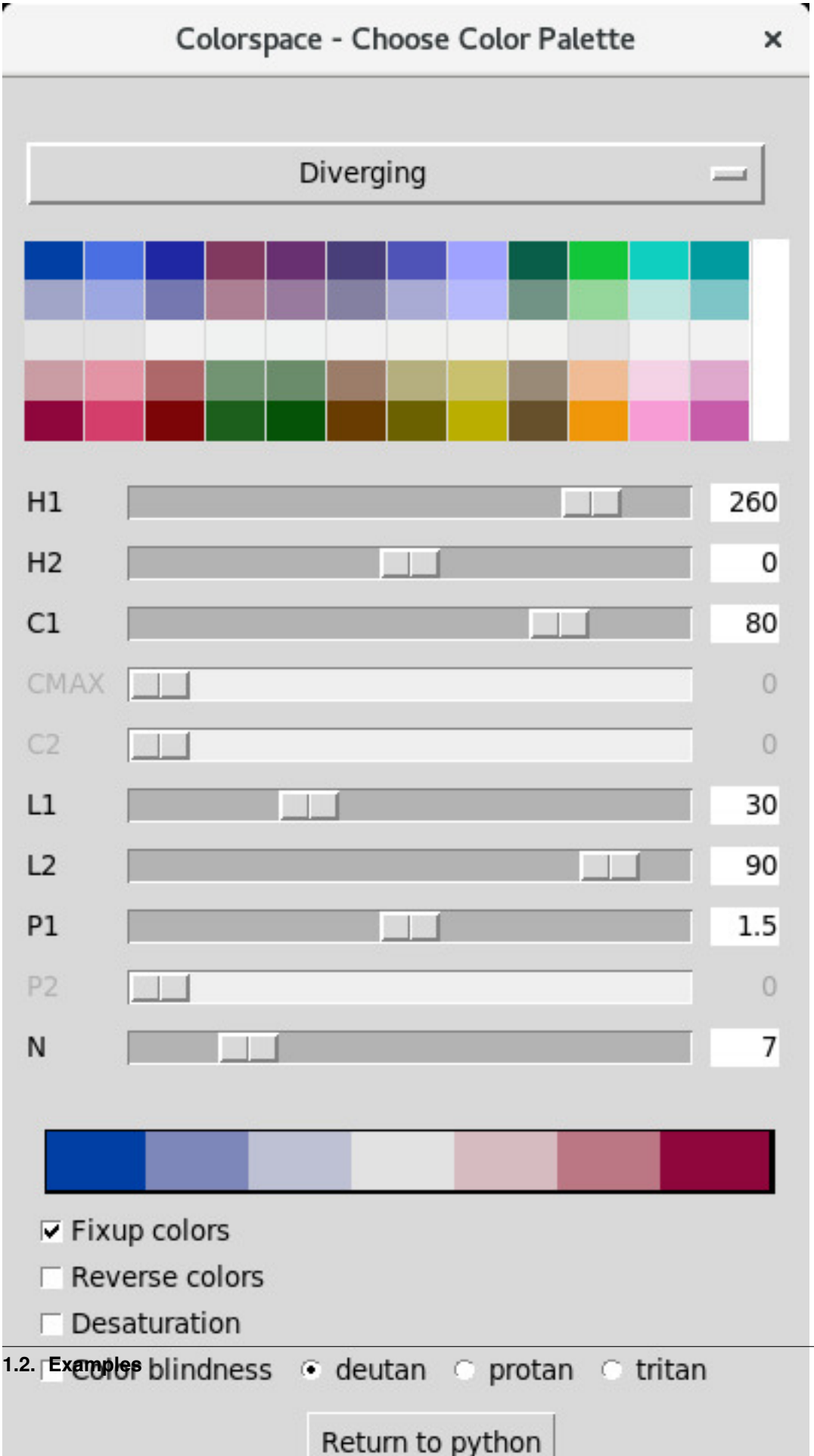
Default Color Palettes

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

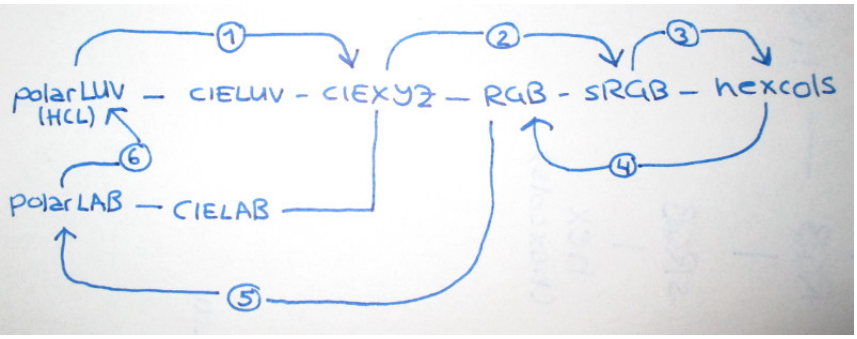
- More information: *Graphical User Interface*

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

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*)

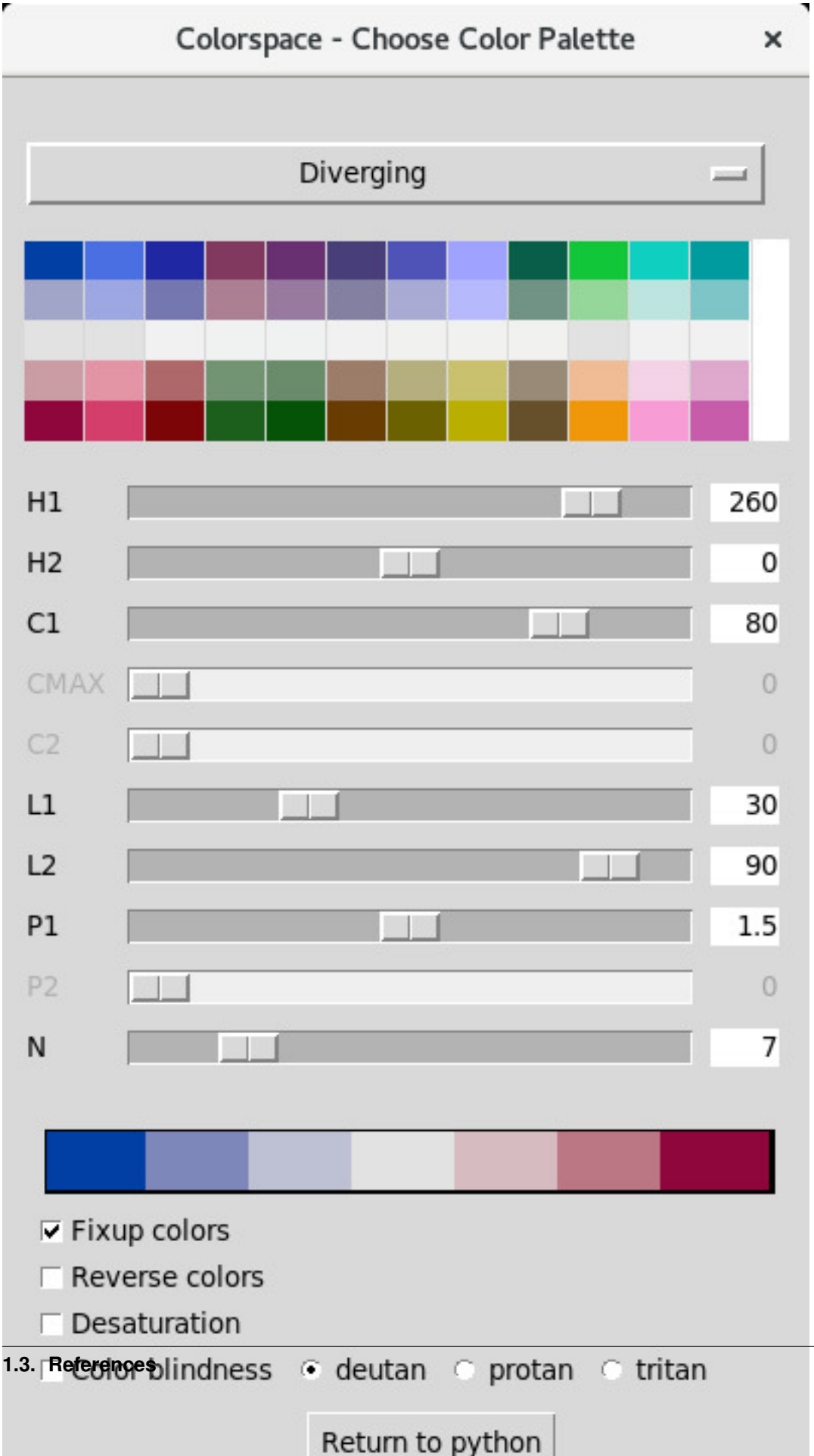
Returns the colors of the current color palette.

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

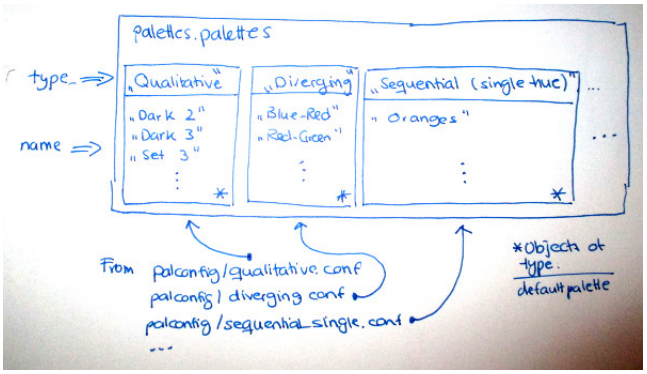
The *colorspace* package contains a set of default palettes which can be accessed via palette name.

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)
```

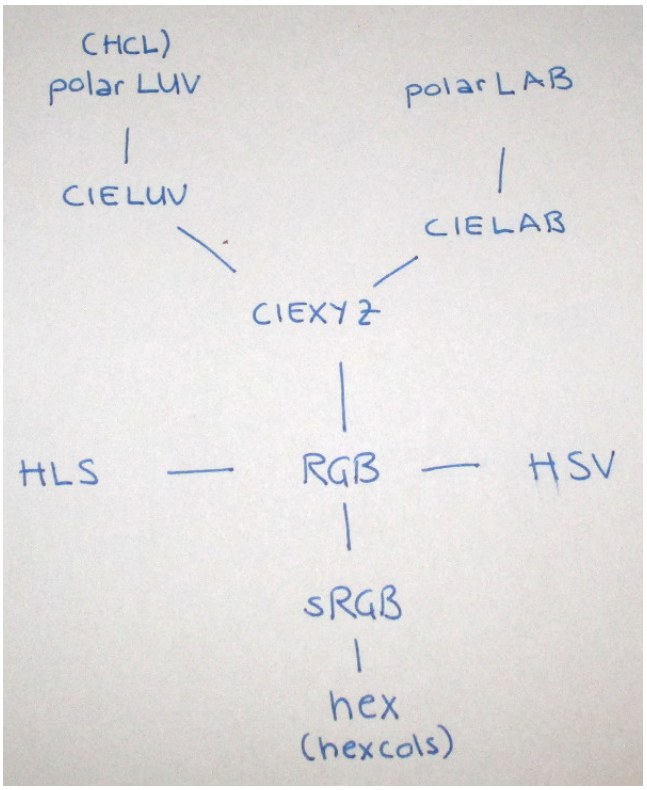
1.4.2 Other Methods

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.

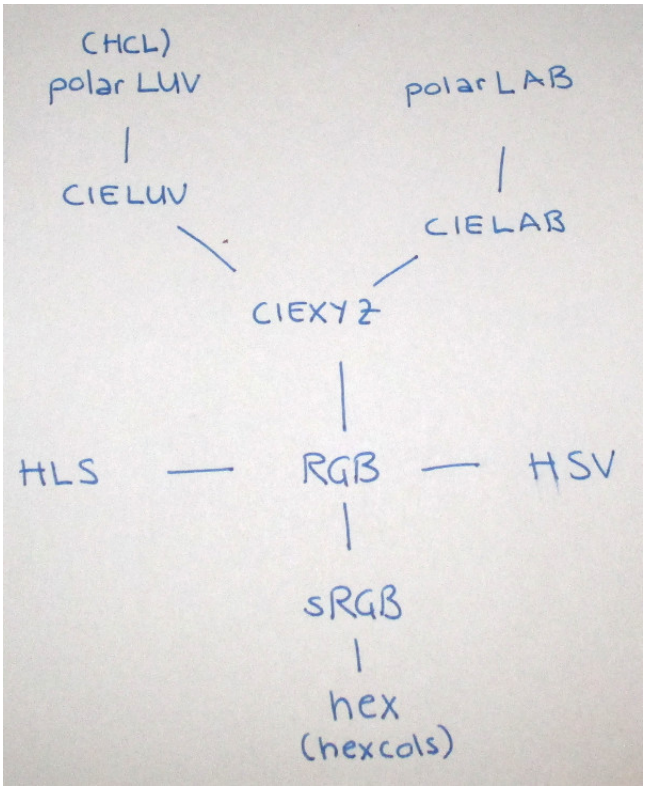
1.4.3 The color objects

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.



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.



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: To be done . . .

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-colorspace/checkouts/stable/docs/hclcolorspace.rst, line 58.)

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/stable/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/stable/colorspace/specplot.py:docstring of specplot.specplot, line 31.)

Python Module Index

C

S

Index