The farbe package

Josef Friedrich
josef@friedrich.rocks
github.com/Josef-Friedrich/farbe

0.1.0 from 2025/05/31

Contents

1 Introduction

Color management (conversion, names) for LuaTEX implemented in Lua.

farbe is mainly a Lua library for converting and manipulating colors. It is based on Lua module lua-color.

CTAN provides an overview page on the subject of Colour: packages to typesetting in colour.

2 Lua interface / API

2.1 Class "Color"

2.1.1 Fields

2.1.1.1 **Color.r** Red component.

2.1.1.2 **Color.g** Green component.

2.1.1.3 **Color.b** Blue component.

2.1.1.4 **Color.a** Alpha component.

2.1.2 Methods

2.1.2.1 **Color:clone ()** Clone color

2.1.2.2 **Color:set (value)** Set color to value.

2.1.2.3 **Color:rgb ()** Get rgb values.

2.1.2.4 **Color:rgba ()** Get rgba values.

2.1.2.5 **Color:hsv ()** Get hsv values.

2.1.2.6 **Color:hsva ()** Get hsv values.

2.1.2.7 **Color:hsl ()** Get hsl values.

2.1.2.8 **Color:hsla ()** Get hsl values.

2.1.2.9 **Color:hwb ()** Get hwb values.

2.1.2.10 **Color:hwba ()** Get hwb values.

2.1.2.11 **Color:cmyk ()** Get cmyk values.

2.1.2.12 **Color:rotate (value)** Rotate hue of color.

2.1.2.13 Color:invert () Invert the color.

2.1.2.14 Color:grey () Reduce saturation to 0.

2.1.2.15 Color:blackOrWhite (lightness) Set to black or white depending on lightness.

2.1.2.16 Color:mix (other, strength) Mix two colors together.

2.1.2.17 Color:complement () Generate complementary color.

2.1.2.18 Color:analogous () Generate analogous color scheme.

2.1.2.19 Color:triad () Generate triadic color scheme.

2.1.2.20 Color:tetrad () Generate tetradic color scheme.

2.1.2.21 Color:compound () Generate compound color scheme.

2.1.2.22 Color:evenlySpaced (n, r) Generate evenly spaced color scheme.

2.1.2.23 Color:tostring (format) Get string representation of color.

2.1.2.24 Color:band (a, b) Apply rgb mask to color, providing backwards compatibility for Lua 5.1 and LuaJIT 2.1.0-beta3

2.1.2.25 Color:isColor (color) Check whether color is a Color.

3 Color names

3.1 base

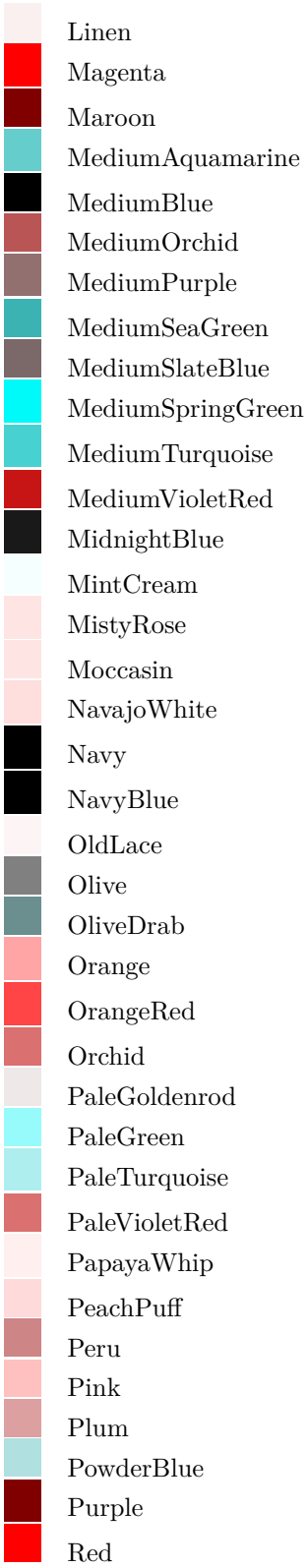
black
blue
brown
cyan
darkgray
gray
green
lightgray
lime
magenta
olive
orange
pink
purple
red
teal
violet
white
yellow

3.2 svg

AliceBlue
AntiqueWhite
Aqua
Aquamarine
Azure
Beige
Bisque
Black
BlanchedAlmond
Blue
BlueViolet
Brown
BurlyWood


CadetBlue
Chartreuse
Chocolate
Coral
CornflowerBlue
Cornsilk
Crimson
Cyan
DarkBlue
DarkCyan
DarkGoldenrod
DarkGray
DarkGreen
DarkGrey
DarkKhaki
DarkMagenta
DarkOliveGreen
DarkOrange
DarkOrchid
DarkRed
DarkSalmon
DarkSeaGreen
DarkSlateBlue
DarkSlateGray
DarkSlateGrey
DarkTurquoise
DarkViolet
DeepPink
DeepSkyBlue
DimGray
DimGrey
DodgerBlue
FireBrick
FloralWhite
ForestGreen
Fuchsia
Gainsboro


GhostWhite
Gold
Goldenrod
Gray
Green
GreenYellow
Grey
Honeydew
HotPink
IndianRed
Indigo
Ivory
Khaki
Lavender
LavenderBlush
LawnGreen
LemonChiffon
LightBlue
LightCoral
LightCyan
LightGoldenrod
LightGoldenrodYellow
LightGray
LightGreen
LightGrey
LightPink
LightSalmon
LightSeaGreen
LightSkyBlue
LightSlateBlue
LightSlateGray
LightSlateGrey
LightSteelBlue
LightYellow
Lime
LimeGreen


Linen
Magenta
Maroon
MediumAquamarine
MediumBlue
MediumOrchid
MediumPurple
MediumSeaGreen
MediumSlateBlue
MediumSpringGreen
MediumTurquoise
MediumVioletRed
MidnightBlue
MintCream
MistyRose
Moccasin
NavajoWhite
Navy
NavyBlue
OldLace
Olive
OliveDrab
Orange
OrangeRed
Orchid
PaleGoldenrod
PaleGreen
PaleTurquoise
PaleVioletRed
PapayaWhip
PeachPuff
Peru
Pink
Plum
PowderBlue
Purple
Red

RosyBrown
RoyalBlue
SaddleBrown
Salmon
SandyBrown
SeaGreen
Seashell
Sienna
Silver
SkyBlue
SlateBlue
SlateGray
SlateGrey
Snow
SpringGreen
SteelBlue
Tan
Teal
Thistle
Tomato
Turquoise
Violet
VioletRed
Wheat
White
WhiteSmoke
Yellow
YellowGreen

3.3 x11

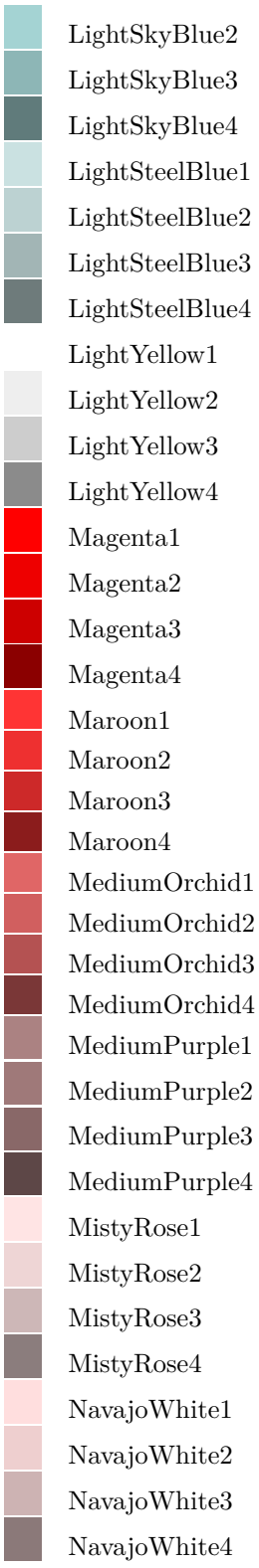
AntiqueWhite1
AntiqueWhite2
AntiqueWhite3
AntiqueWhite4
Aquamarine1
Aquamarine2
Aquamarine3


Aquamarine4
Azure1
Azure2
Azure3
Azure4
Bisque1
Bisque2
Bisque3
Bisque4
Blue1
Blue2
Blue3
Blue4
Brown1
Brown2
Brown3
Brown4
Burlywood1
Burlywood2
Burlywood3
Burlywood4
CadetBlue1
CadetBlue2
CadetBlue3
CadetBlue4
Chartreuse1
Chartreuse2
Chartreuse3
Chartreuse4
Chocolate1
Chocolate2
Chocolate3
Chocolate4
Coral1
Coral2
Coral3
Coral4


Cornsilk1
Cornsilk2
Cornsilk3
Cornsilk4
Cyan1
Cyan2
Cyan3
Cyan4
DarkGoldenrod1
DarkGoldenrod2
DarkGoldenrod3
DarkGoldenrod4
DarkOliveGreen1
DarkOliveGreen2
DarkOliveGreen3
DarkOliveGreen4
DarkOrange1
DarkOrange2
DarkOrange3
DarkOrange4
DarkOrchid1
DarkOrchid2
DarkOrchid3
DarkOrchid4
DarkSeaGreen1
DarkSeaGreen2
DarkSeaGreen3
DarkSeaGreen4
DarkSlateGray1
DarkSlateGray2
DarkSlateGray3
DarkSlateGray4
DeepPink1
DeepPink2
DeepPink3
DeepPink4


DeepSkyBlue1
DeepSkyBlue2
DeepSkyBlue3
DeepSkyBlue4
DodgerBlue1
DodgerBlue2
DodgerBlue3
DodgerBlue4
Firebrick1
Firebrick2
Firebrick3
Firebrick4
Gold1
Gold2
Gold3
Gold4
Goldenrod1
Goldenrod2
Goldenrod3
Goldenrod4
Green1
Green2
Green3
Green4
Honeydew1
Honeydew2
Honeydew3
Honeydew4
HotPink1
HotPink2
HotPink3
HotPink4
IndianRed1
IndianRed2
IndianRed3
IndianRed4
Ivory1


Ivory2
Ivory3
Ivory4
Khaki1
Khaki2
Khaki3
Khaki4
LavenderBlush1
LavenderBlush2
LavenderBlush3
LavenderBlush4
LemonChiffon1
LemonChiffon2
LemonChiffon3
LemonChiffon4
LightBlue1
LightBlue2
LightBlue3
LightBlue4
LightCyan1
LightCyan2
LightCyan3
LightCyan4
LightGoldenrod1
LightGoldenrod2
LightGoldenrod3
LightGoldenrod4
LightPink1
LightPink2
LightPink3
LightPink4
LightSalmon1
LightSalmon2
LightSalmon3
LightSalmon4
LightSkyBlue1


LightSkyBlue2
LightSkyBlue3
LightSkyBlue4
LightSteelBlue1
LightSteelBlue2
LightSteelBlue3
LightSteelBlue4
LightYellow1
LightYellow2
LightYellow3
LightYellow4
Magenta1
Magenta2
Magenta3
Magenta4
Maroon1
Maroon2
Maroon3
Maroon4
MediumOrchid1
MediumOrchid2
MediumOrchid3
MediumOrchid4
MediumPurple1
MediumPurple2
MediumPurple3
MediumPurple4
MistyRose1
MistyRose2
MistyRose3
MistyRose4
NavajoWhite1
NavajoWhite2
NavajoWhite3
NavajoWhite4


OliveDrab1
OliveDrab2
OliveDrab3
OliveDrab4
Orange1
Orange2
Orange3
Orange4
OrangeRed1
OrangeRed2
OrangeRed3
OrangeRed4
Orchid1
Orchid2
Orchid3
Orchid4
PaleGreen1
PaleGreen2
PaleGreen3
PaleGreen4
PaleTurquoise1
PaleTurquoise2
PaleTurquoise3
PaleTurquoise4
PaleVioletRed1
PaleVioletRed2
PaleVioletRed3
PaleVioletRed4
PeachPuff1
PeachPuff2
PeachPuff3
PeachPuff4
Pink1
Pink2
Pink3
Pink4
Plum1


Plum2
Plum3
Plum4
Purple1
Purple2
Purple3
Purple4
Red1
Red2
Red3
Red4
RosyBrown1
RosyBrown2
RosyBrown3
RosyBrown4
RoyalBlue1
RoyalBlue2
RoyalBlue3
RoyalBlue4
Salmon1
Salmon2
Salmon3
Salmon4
SeaGreen1
SeaGreen2
SeaGreen3
SeaGreen4
Seashell1
Seashell2
Seashell3
Seashell4
Sienna1
Sienna2
Sienna3
Sienna4
SkyBlue1
SkyBlue2


SkyBlue3
SkyBlue4
SlateBlue1
SlateBlue2
SlateBlue3
SlateBlue4
SlateGray1
SlateGray2
SlateGray3
SlateGray4
Snow1
Snow2
Snow3
Snow4
SpringGreen1
SpringGreen2
SpringGreen3
SpringGreen4
SteelBlue1
SteelBlue2
SteelBlue3
SteelBlue4
Tan1
Tan2
Tan3
Tan4
Thistle1
Thistle2
Thistle3
Thistle4
Tomato1
Tomato2
Tomato3
Tomato4
Turquoise1
Turquoise2
Turquoise3


Turquoise4
VioletRed1
VioletRed2
VioletRed3
VioletRed4
Wheat1
Wheat2
Wheat3
Wheat4
Yellow1
Yellow2
Yellow3
Yellow4
Gray0
Green0
Grey0
Maroon0
Purple0

4 Implementation

4.1 farbe.lua

```lua
-- farbe.lua
-- Copyright 2025 Josef Friedrich
--
-- This work may be distributed and/or modified under the
-- conditions of the LaTeX Project Public License, either version 1.3c
-- of this license or (at your option) any later version.
-- The latest version of this license is in
--   http://www.latex-project.org/lppl.txt
-- and version 1.3c or later is part of all distributions of LaTeX
-- version 2008/05/04 or later.
--
-- This work has the LPPL maintenance status `maintained'.
--
-- The Current Maintainer of this work is Josef Friedrich.
--
-- This work consists of the files farbe.lua, farbe.tex,
-- and farbe.sty.
-- https://github.com/latex3/xcolor/blob/main/xcolor.dtx
---@alias r number red (0.0 - 1.0)
---@alias g number green (0.0 - 1.0)
---@alias b number blue (0.0 - 1.0)
---@alias a number alpha (0.0 - 1.0)
---@alias c number cyan (0.0 - 1.0)
---@alias m number magenta (0.0 - 1.0)
---@alias y number yellow (0.0 - 1.0)
---@alias k number key(black) (0.0 - 1.0)
local schemes = {

  base = {
    'black',
    'blue',
    'brown',
    'cyan',
    'darkgray',
    'gray',
    'green',
    'lightgray',
    'lime',
    'magenta',
    'olive',
    'orange',
    'pink',
    'purple',
    'red',
    'teal',
    'violet',
    'white',
    'yellow',
  },

  svg = {
    'AliceBlue',
    'AntiqueWhite',
    'Aqua',
    'Aquamarine',
    'Azure',
    'Beige',
    'Bisque',
    'Black',
    'BlanchedAlmond',
```

```
61      'Blue',
62      'BlueViolet',
63      'Brown',
64      'BurlyWood',
65      'CadetBlue',
66      'Chartreuse',
67      'Chocolate',
68      'Coral',
69      'CornflowerBlue',
70      'Cornsilk',
71      'Crimson',
72      'Cyan',
73      'DarkBlue',
74      'DarkCyan',
75      'DarkGoldenrod',
76      'DarkGray',
77      'DarkGreen',
78      'DarkGrey',
79      'DarkKhaki',
80      'DarkMagenta',
81      'DarkOliveGreen',
82      'DarkOrange',
83      'DarkOrchid',
84      'DarkRed',
85      'DarkSalmon',
86      'DarkSeaGreen',
87      'DarkSlateBlue',
88      'DarkSlateGray',
89      'DarkSlateGrey',
90      'DarkTurquoise',
91      'DarkViolet',
92      'DeepPink',
93      'DeepSkyBlue',
94      'DimGray',
95      'DimGrey',
96      'DodgerBlue',
97      'FireBrick',
98      'FloralWhite',
99      'ForestGreen',
100     'Fuchsia',
101     'Gainsboro',
102     'GhostWhite',
103     'Gold',
104     'Goldenrod',
105     'Gray',
106     'Green',
107     'GreenYellow',
108     'Grey',
109     'Honeydew',
110     'HotPink',
111     'IndianRed',
112     'Indigo',
113     'Ivory',
114     'Khaki',
115     'Lavender',
116     'LavenderBlush',
117     'LawnGreen',
118     'LemonChiffon',
119     'LightBlue',
120     'LightCoral',
121     'LightCyan',
122     'LightGoldenrod',
123     'LightGoldenrodYellow',
124     'LightGray',
```

```
125         'LightGreen',
126         'LightGrey',
127         'LightPink',
128         'LightSalmon',
129         'LightSeaGreen',
130         'LightSkyBlue',
131         'LightSlateBlue',
132         'LightSlateGray',
133         'LightSlateGrey',
134         'LightSteelBlue',
135         'LightYellow',
136         'Lime',
137         'LimeGreen',
138         'Linen',
139         'Magenta',
140         'Maroon',
141         'MediumAquamarine',
142         'MediumBlue',
143         'MediumOrchid',
144         'MediumPurple',
145         'MediumSeaGreen',
146         'MediumSlateBlue',
147         'MediumSpringGreen',
148         'MediumTurquoise',
149         'MediumVioletRed',
150         'MidnightBlue',
151         'MintCream',
152         'MistyRose',
153         'Moccasin',
154         'NavajoWhite',
155         'Navy',
156         'NavyBlue',
157         'OldLace',
158         'Olive',
159         'OliveDrab',
160         'Orange',
161         'OrangeRed',
162         'Orchid',
163         'PaleGoldenrod',
164         'PaleGreen',
165         'PaleTurquoise',
166         'PaleVioletRed',
167         'PapayaWhip',
168         'PeachPuff',
169         'Peru',
170         'Pink',
171         'Plum',
172         'PowderBlue',
173         'Purple',
174         'Red',
175         'RosyBrown',
176         'RoyalBlue',
177         'SaddleBrown',
178         'Salmon',
179         'SandyBrown',
180         'SeaGreen',
181         'Seashell',
182         'Sienna',
183         'Silver',
184         'SkyBlue',
185         'SlateBlue',
186         'SlateGray',
187         'SlateGrey',
188         'Snow',
```

```
    'SpringGreen',
    'SteelBlue',
    'Tan',
    'Teal',
    'Thistle',
    'Tomato',
    'Turquoise',
    'Violet',
    'VioletRed',
    'Wheat',
    'White',
    'WhiteSmoke',
    'Yellow',
    'YellowGreen',
  },

  x11 = {
    'AntiqueWhite1',
    'AntiqueWhite2',
    'AntiqueWhite3',
    'AntiqueWhite4',
    'Aquamarine1',
    'Aquamarine2',
    'Aquamarine3',
    'Aquamarine4',
    'Azure1',
    'Azure2',
    'Azure3',
    'Azure4',
    'Bisque1',
    'Bisque2',
    'Bisque3',
    'Bisque4',
    'Blue1',
    'Blue2',
    'Blue3',
    'Blue4',
    'Brown1',
    'Brown2',
    'Brown3',
    'Brown4',
    'Burlywood1',
    'Burlywood2',
    'Burlywood3',
    'Burlywood4',
    'CadetBlue1',
    'CadetBlue2',
    'CadetBlue3',
    'CadetBlue4',
    'Chartreuse1',
    'Chartreuse2',
    'Chartreuse3',
    'Chartreuse4',
    'Chocolate1',
    'Chocolate2',
    'Chocolate3',
    'Chocolate4',
    'Coral1',
    'Coral2',
    'Coral3',
    'Coral4',
    'Cornsilk1',
    'Cornsilk2',
    'Cornsilk3',
```

```
    'Cornsilk4',
    'Cyan1',
    'Cyan2',
    'Cyan3',
    'Cyan4',
    'DarkGoldenrod1',
    'DarkGoldenrod2',
    'DarkGoldenrod3',
    'DarkGoldenrod4',
    'DarkOliveGreen1',
    'DarkOliveGreen2',
    'DarkOliveGreen3',
    'DarkOliveGreen4',
    'DarkOrange1',
    'DarkOrange2',
    'DarkOrange3',
    'DarkOrange4',
    'DarkOrchid1',
    'DarkOrchid2',
    'DarkOrchid3',
    'DarkOrchid4',
    'DarkSeaGreen1',
    'DarkSeaGreen2',
    'DarkSeaGreen3',
    'DarkSeaGreen4',
    'DarkSlateGray1',
    'DarkSlateGray2',
    'DarkSlateGray3',
    'DarkSlateGray4',
    'DeepPink1',
    'DeepPink2',
    'DeepPink3',
    'DeepPink4',
    'DeepSkyBlue1',
    'DeepSkyBlue2',
    'DeepSkyBlue3',
    'DeepSkyBlue4',
    'DodgerBlue1',
    'DodgerBlue2',
    'DodgerBlue3',
    'DodgerBlue4',
    'Firebrick1',
    'Firebrick2',
    'Firebrick3',
    'Firebrick4',
    'Gold1',
    'Gold2',
    'Gold3',
    'Gold4',
    'Goldenrod1',
    'Goldenrod2',
    'Goldenrod3',
    'Goldenrod4',
    'Green1',
    'Green2',
    'Green3',
    'Green4',
    'Honeydew1',
    'Honeydew2',
    'Honeydew3',
    'Honeydew4',
    'HotPink1',
    'HotPink2',
    'HotPink3',
```

```
        'HotPink4',
        'IndianRed1',
        'IndianRed2',
        'IndianRed3',
        'IndianRed4',
        'Ivory1',
        'Ivory2',
        'Ivory3',
        'Ivory4',
        'Khaki1',
        'Khaki2',
        'Khaki3',
        'Khaki4',
        'LavenderBlush1',
        'LavenderBlush2',
        'LavenderBlush3',
        'LavenderBlush4',
        'LemonChiffon1',
        'LemonChiffon2',
        'LemonChiffon3',
        'LemonChiffon4',
        'LightBlue1',
        'LightBlue2',
        'LightBlue3',
        'LightBlue4',
        'LightCyan1',
        'LightCyan2',
        'LightCyan3',
        'LightCyan4',
        'LightGoldenrod1',
        'LightGoldenrod2',
        'LightGoldenrod3',
        'LightGoldenrod4',
        'LightPink1',
        'LightPink2',
        'LightPink3',
        'LightPink4',
        'LightSalmon1',
        'LightSalmon2',
        'LightSalmon3',
        'LightSalmon4',
        'LightSkyBlue1',
        'LightSkyBlue2',
        'LightSkyBlue3',
        'LightSkyBlue4',
        'LightSteelBlue1',
        'LightSteelBlue2',
        'LightSteelBlue3',
        'LightSteelBlue4',
        'LightYellow1',
        'LightYellow2',
        'LightYellow3',
        'LightYellow4',
        'Magenta1',
        'Magenta2',
        'Magenta3',
        'Magenta4',
        'Maroon1',
        'Maroon2',
        'Maroon3',
        'Maroon4',
        'MediumOrchid1',
        'MediumOrchid2',
        'MediumOrchid3',
```

```
        'MediumOrchid4',
        'MediumPurple1',
        'MediumPurple2',
        'MediumPurple3',
        'MediumPurple4',
        'MistyRose1',
        'MistyRose2',
        'MistyRose3',
        'MistyRose4',
        'NavajoWhite1',
        'NavajoWhite2',
        'NavajoWhite3',
        'NavajoWhite4',
        'OliveDrab1',
        'OliveDrab2',
        'OliveDrab3',
        'OliveDrab4',
        'Orange1',
        'Orange2',
        'Orange3',
        'Orange4',
        'OrangeRed1',
        'OrangeRed2',
        'OrangeRed3',
        'OrangeRed4',
        'Orchid1',
        'Orchid2',
        'Orchid3',
        'Orchid4',
        'PaleGreen1',
        'PaleGreen2',
        'PaleGreen3',
        'PaleGreen4',
        'PaleTurquoise1',
        'PaleTurquoise2',
        'PaleTurquoise3',
        'PaleTurquoise4',
        'PaleVioletRed1',
        'PaleVioletRed2',
        'PaleVioletRed3',
        'PaleVioletRed4',
        'PeachPuff1',
        'PeachPuff2',
        'PeachPuff3',
        'PeachPuff4',
        'Pink1',
        'Pink2',
        'Pink3',
        'Pink4',
        'Plum1',
        'Plum2',
        'Plum3',
        'Plum4',
        'Purple1',
        'Purple2',
        'Purple3',
        'Purple4',
        'Red1',
        'Red2',
        'Red3',
        'Red4',
        'RosyBrown1',
        'RosyBrown2',
        'RosyBrown3',
```

```
        'RosyBrown4',
        'RoyalBlue1',
        'RoyalBlue2',
        'RoyalBlue3',
        'RoyalBlue4',
        'Salmon1',
        'Salmon2',
        'Salmon3',
        'Salmon4',
        'SeaGreen1',
        'SeaGreen2',
        'SeaGreen3',
        'SeaGreen4',
        'Seashell1',
        'Seashell2',
        'Seashell3',
        'Seashell4',
        'Sienna1',
        'Sienna2',
        'Sienna3',
        'Sienna4',
        'SkyBlue1',
        'SkyBlue2',
        'SkyBlue3',
        'SkyBlue4',
        'SlateBlue1',
        'SlateBlue2',
        'SlateBlue3',
        'SlateBlue4',
        'SlateGray1',
        'SlateGray2',
        'SlateGray3',
        'SlateGray4',
        'Snow1',
        'Snow2',
        'Snow3',
        'Snow4',
        'SpringGreen1',
        'SpringGreen2',
        'SpringGreen3',
        'SpringGreen4',
        'SteelBlue1',
        'SteelBlue2',
        'SteelBlue3',
        'SteelBlue4',
        'Tan1',
        'Tan2',
        'Tan3',
        'Tan4',
        'Thistle1',
        'Thistle2',
        'Thistle3',
        'Thistle4',
        'Tomato1',
        'Tomato2',
        'Tomato3',
        'Tomato4',
        'Turquoise1',
        'Turquoise2',
        'Turquoise3',
        'Turquoise4',
        'VioletRed1',
        'VioletRed2',
        'VioletRed3',
```

```
    'VioletRed4',
    'Wheat1',
    'Wheat2',
    'Wheat3',
    'Wheat4',
    'Yellow1',
    'Yellow2',
    'Yellow3',
    'Yellow4',
    'Gray0',
    'Green0',
    'Grey0',
    'Maroon0',
    'Purple0',
  },
}

local log = (function()
  local opts = { verbosity = 0 }

  local function print_message(message, ...)
    print(string.format(message, ...))
  end

  local function info(message, ...)
    if opts.verbosity > 0 then
      print_message(message, ...)
    end
  end

  local function debug(message, ...)
    if opts.verbosity > 1 then
      print_message(message, ...)
    end
  end

  local function verbose(message, ...)
    if opts.verbosity > 2 then
      print_message(message, ...)
    end
  end

  return { opts = opts, info = info, debug = debug, verbose = verbose }
end)()

log.opts.verbosity = 3

local colors = {
  -- base
  black = { 0, 0, 0 },
  blue = { 0, 0, 1 },
  brown = { 0.75, 0.5, 0.25 },
  cyan = { 0, 1, 1 },
  darkgray = { 0.25, 0.25, 0.25 },
  gray = { 0.5, 0.5, 0.5 },
  green = { 0, 1, 0 },
  lightgray = { 0.75, 0.75, 0.75 },
  lime = { 0.75, 1, 0 },
  magenta = { 1, 0, 1 },
  olive = { 0.5, 0.5, 0 },
  orange = { 1, 0.5, 0 },
  pink = { 1, 0.75, 0.75 },
  purple = { 0.75, 0, 0.25 },
  red = { 1, 0, 0 },
```

```
    teal = { 0, 0.5, 0.5 },
    violet = { 0.5, 0, 0.5 },
    white = { 1, 1, 1 },
    yellow = { 1, 1, 0 },

    -- svg
    AliceBlue = { 0.94, 0.972, 0.972 },
    AntiqueWhite = { 0.98, 0.92, 0.92 },
    Aqua = { 0, 1, 1 },
    Aquamarine = { 0.498, 1, 1 },
    Azure = { 0.94, 1, 1 },
    Beige = { 0.96, 0.96, 0.96 },
    Bisque = { 1, 0.894, 0.894 },
    Black = { 0, 0, 0 },
    BlanchedAlmond = { 1, 0.92, 0.92 },
    Blue = { 0, 0, 0 },
    BlueViolet = { 0.54, 0.17, 0.17 },
    Brown = { 0.648, 0.165, 0.165 },
    BurlyWood = { 0.87, 0.72, 0.72 },
    CadetBlue = { 0.372, 0.62, 0.62 },
    Chartreuse = { 0.498, 1, 1 },
    Chocolate = { 0.824, 0.41, 0.41 },
    Coral = { 1, 0.498, 0.498 },
    CornflowerBlue = { 0.392, 0.585, 0.585 },
    Cornsilk = { 1, 0.972, 0.972 },
    Crimson = { 0.864, 0.08, 0.08 },
    Cyan = { 0, 1, 1 },
    DarkBlue = { 0, 0, 0 },
    DarkCyan = { 0, 0.545, 0.545 },
    DarkGoldenrod = { 0.72, 0.525, 0.525 },
    DarkGray = { 0.664, 0.664, 0.664 },
    DarkGreen = { 0, 0.392, 0.392 },
    DarkGrey = { 0.664, 0.664, 0.664 },
    DarkKhaki = { 0.74, 0.716, 0.716 },
    DarkMagenta = { 0.545, 0, 0 },
    DarkOliveGreen = { 0.332, 0.42, 0.42 },
    DarkOrange = { 1, 0.55, 0.55 },
    DarkOrchid = { 0.6, 0.196, 0.196 },
    DarkRed = { 0.545, 0, 0 },
    DarkSalmon = { 0.912, 0.59, 0.59 },
    DarkSeaGreen = { 0.56, 0.736, 0.736 },
    DarkSlateBlue = { 0.284, 0.24, 0.24 },
    DarkSlateGray = { 0.185, 0.31, 0.31 },
    DarkSlateGrey = { 0.185, 0.31, 0.31 },
    DarkTurquoise = { 0, 0.808, 0.808 },
    DarkViolet = { 0.58, 0, 0 },
    DeepPink = { 1, 0.08, 0.08 },
    DeepSkyBlue = { 0, 0.75, 0.75 },
    DimGray = { 0.41, 0.41, 0.41 },
    DimGrey = { 0.41, 0.41, 0.41 },
    DodgerBlue = { 0.116, 0.565, 0.565 },
    FireBrick = { 0.698, 0.132, 0.132 },
    FloralWhite = { 1, 0.98, 0.98 },
    ForestGreen = { 0.132, 0.545, 0.545 },
    Fuchsia = { 1, 0, 0 },
    Gainsboro = { 0.864, 0.864, 0.864 },
    GhostWhite = { 0.972, 0.972, 0.972 },
    Gold = { 1, 0.844, 0.844 },
    Goldenrod = { 0.855, 0.648, 0.648 },
    Gray = { 0.5, 0.5, 0.5 },
    Green = { 0, 0.5, 0.5 },
    GreenYellow = { 0.68, 1, 1 },
    Grey = { 0.5, 0.5, 0.5 },
    Honeydew = { 0.94, 1, 1 },
```

```
    HotPink = { 1, 0.41, 0.41 },
    IndianRed = { 0.804, 0.36, 0.36 },
    Indigo = { 0.294, 0, 0 },
    Ivory = { 1, 1, 1 },
    Khaki = { 0.94, 0.9, 0.9 },
    Lavender = { 0.9, 0.9, 0.9 },
    LavenderBlush = { 1, 0.94, 0.94 },
    LawnGreen = { 0.488, 0.99, 0.99 },
    LemonChiffon = { 1, 0.98, 0.98 },
    LightBlue = { 0.68, 0.848, 0.848 },
    LightCoral = { 0.94, 0.5, 0.5 },
    LightCyan = { 0.88, 1, 1 },
    LightGoldenrod = { 0.933, 0.867, 0.867 },
    LightGoldenrodYellow = { 0.98, 0.98, 0.98 },
    LightGray = { 0.828, 0.828, 0.828 },
    LightGreen = { 0.565, 0.932, 0.932 },
    LightGrey = { 0.828, 0.828, 0.828 },
    LightPink = { 1, 0.712, 0.712 },
    LightSalmon = { 1, 0.628, 0.628 },
    LightSeaGreen = { 0.125, 0.698, 0.698 },
    LightSkyBlue = { 0.53, 0.808, 0.808 },
    LightSlateBlue = { 0.518, 0.44, 0.44 },
    LightSlateGray = { 0.468, 0.532, 0.532 },
    LightSlateGrey = { 0.468, 0.532, 0.532 },
    LightSteelBlue = { 0.69, 0.77, 0.77 },
    LightYellow = { 1, 1, 1 },
    Lime = { 0, 1, 1 },
    LimeGreen = { 0.196, 0.804, 0.804 },
    Linen = { 0.98, 0.94, 0.94 },
    Magenta = { 1, 0, 0 },
    Maroon = { 0.5, 0, 0 },
    MediumAquamarine = { 0.4, 0.804, 0.804 },
    MediumBlue = { 0, 0, 0 },
    MediumOrchid = { 0.73, 0.332, 0.332 },
    MediumPurple = { 0.576, 0.44, 0.44 },
    MediumSeaGreen = { 0.235, 0.7, 0.7 },
    MediumSlateBlue = { 0.484, 0.408, 0.408 },
    MediumSpringGreen = { 0, 0.98, 0.98 },
    MediumTurquoise = { 0.284, 0.82, 0.82 },
    MediumVioletRed = { 0.78, 0.084, 0.084 },
    MidnightBlue = { 0.098, 0.098, 0.098 },
    MintCream = { 0.96, 1, 1 },
    MistyRose = { 1, 0.894, 0.894 },
    Moccasin = { 1, 0.894, 0.894 },
    NavajoWhite = { 1, 0.87, 0.87 },
    Navy = { 0, 0, 0 },
    NavyBlue = { 0, 0, 0 },
    OldLace = { 0.992, 0.96, 0.96 },
    Olive = { 0.5, 0.5, 0.5 },
    OliveDrab = { 0.42, 0.556, 0.556 },
    Orange = { 1, 0.648, 0.648 },
    OrangeRed = { 1, 0.27, 0.27 },
    Orchid = { 0.855, 0.44, 0.44 },
    PaleGoldenrod = { 0.932, 0.91, 0.91 },
    PaleGreen = { 0.596, 0.985, 0.985 },
    PaleTurquoise = { 0.688, 0.932, 0.932 },
    PaleVioletRed = { 0.86, 0.44, 0.44 },
    PapayaWhip = { 1, 0.936, 0.936 },
    PeachPuff = { 1, 0.855, 0.855 },
    Peru = { 0.804, 0.52, 0.52 },
    Pink = { 1, 0.752, 0.752 },
    Plum = { 0.868, 0.628, 0.628 },
    PowderBlue = { 0.69, 0.88, 0.88 },
    Purple = { 0.5, 0, 0 },
```

```
    Red = { 1, 0, 0 },
    RosyBrown = { 0.736, 0.56, 0.56 },
    RoyalBlue = { 0.255, 0.41, 0.41 },
    SaddleBrown = { 0.545, 0.27, 0.27 },
    Salmon = { 0.98, 0.5, 0.5 },
    SandyBrown = { 0.956, 0.644, 0.644 },
    SeaGreen = { 0.18, 0.545, 0.545 },
    Seashell = { 1, 0.96, 0.96 },
    Sienna = { 0.628, 0.32, 0.32 },
    Silver = { 0.752, 0.752, 0.752 },
    SkyBlue = { 0.53, 0.808, 0.808 },
    SlateBlue = { 0.415, 0.352, 0.352 },
    SlateGray = { 0.44, 0.5, 0.5 },
    SlateGrey = { 0.44, 0.5, 0.5 },
    Snow = { 1, 0.98, 0.98 },
    SpringGreen = { 0, 1, 1 },
    SteelBlue = { 0.275, 0.51, 0.51 },
    Tan = { 0.824, 0.705, 0.705 },
    Teal = { 0, 0.5, 0.5 },
    Thistle = { 0.848, 0.75, 0.75 },
    Tomato = { 1, 0.39, 0.39 },
    Turquoise = { 0.25, 0.88, 0.88 },
    Violet = { 0.932, 0.51, 0.51 },
    VioletRed = { 0.816, 0.125, 0.125 },
    Wheat = { 0.96, 0.87, 0.87 },
    White = { 1, 1, 1 },
    WhiteSmoke = { 0.96, 0.96, 0.96 },
    Yellow = { 1, 1, 1 },
    YellowGreen = { 0.604, 0.804, 0.804 },

    ---x11
    AntiqueWhite1 = { 1, 0.936, 0.936 },
    AntiqueWhite2 = { 0.932, 0.875, 0.875 },
    AntiqueWhite3 = { 0.804, 0.752, 0.752 },
    AntiqueWhite4 = { 0.545, 0.512, 0.512 },
    Aquamarine1 = { 0.498, 1, 1 },
    Aquamarine2 = { 0.464, 0.932, 0.932 },
    Aquamarine3 = { 0.4, 0.804, 0.804 },
    Aquamarine4 = { 0.27, 0.545, 0.545 },
    Azure1 = { 0.94, 1, 1 },
    Azure2 = { 0.88, 0.932, 0.932 },
    Azure3 = { 0.756, 0.804, 0.804 },
    Azure4 = { 0.512, 0.545, 0.545 },
    Bisque1 = { 1, 0.894, 0.894 },
    Bisque2 = { 0.932, 0.835, 0.835 },
    Bisque3 = { 0.804, 0.716, 0.716 },
    Bisque4 = { 0.545, 0.49, 0.49 },
    Blue1 = { 0, 0, 0 },
    Blue2 = { 0, 0, 0 },
    Blue3 = { 0, 0, 0 },
    Blue4 = { 0, 0, 0 },
    Brown1 = { 1, 0.25, 0.25 },
    Brown2 = { 0.932, 0.23, 0.23 },
    Brown3 = { 0.804, 0.2, 0.2 },
    Brown4 = { 0.545, 0.136, 0.136 },
    Burlywood1 = { 1, 0.828, 0.828 },
    Burlywood2 = { 0.932, 0.772, 0.772 },
    Burlywood3 = { 0.804, 0.668, 0.668 },
    Burlywood4 = { 0.545, 0.45, 0.45 },
    CadetBlue1 = { 0.596, 0.96, 0.96 },
    CadetBlue2 = { 0.556, 0.898, 0.898 },
    CadetBlue3 = { 0.48, 0.772, 0.772 },
    CadetBlue4 = { 0.325, 0.525, 0.525 },
    Chartreuse1 = { 0.498, 1, 1 },
```

```
    Chartreuse2 = { 0.464, 0.932, 0.932 },
    Chartreuse3 = { 0.4, 0.804, 0.804 },
    Chartreuse4 = { 0.27, 0.545, 0.545 },
    Chocolate1 = { 1, 0.498, 0.498 },
    Chocolate2 = { 0.932, 0.464, 0.464 },
    Chocolate3 = { 0.804, 0.4, 0.4 },
    Chocolate4 = { 0.545, 0.27, 0.27 },
    Coral1 = { 1, 0.448, 0.448 },
    Coral2 = { 0.932, 0.415, 0.415 },
    Coral3 = { 0.804, 0.356, 0.356 },
    Coral4 = { 0.545, 0.244, 0.244 },
    Cornsilk1 = { 1, 0.972, 0.972 },
    Cornsilk2 = { 0.932, 0.91, 0.91 },
    Cornsilk3 = { 0.804, 0.785, 0.785 },
    Cornsilk4 = { 0.545, 0.532, 0.532 },
    Cyan1 = { 0, 1, 1 },
    Cyan2 = { 0, 0.932, 0.932 },
    Cyan3 = { 0, 0.804, 0.804 },
    Cyan4 = { 0, 0.545, 0.545 },
    DarkGoldenrod1 = { 1, 0.725, 0.725 },
    DarkGoldenrod2 = { 0.932, 0.68, 0.68 },
    DarkGoldenrod3 = { 0.804, 0.585, 0.585 },
    DarkGoldenrod4 = { 0.545, 0.396, 0.396 },
    DarkOliveGreen1 = { 0.792, 1, 1 },
    DarkOliveGreen2 = { 0.736, 0.932, 0.932 },
    DarkOliveGreen3 = { 0.635, 0.804, 0.804 },
    DarkOliveGreen4 = { 0.43, 0.545, 0.545 },
    DarkOrange1 = { 1, 0.498, 0.498 },
    DarkOrange2 = { 0.932, 0.464, 0.464 },
    DarkOrange3 = { 0.804, 0.4, 0.4 },
    DarkOrange4 = { 0.545, 0.27, 0.27 },
    DarkOrchid1 = { 0.75, 0.244, 0.244 },
    DarkOrchid2 = { 0.698, 0.228, 0.228 },
    DarkOrchid3 = { 0.604, 0.196, 0.196 },
    DarkOrchid4 = { 0.408, 0.132, 0.132 },
    DarkSeaGreen1 = { 0.756, 1, 1 },
    DarkSeaGreen2 = { 0.705, 0.932, 0.932 },
    DarkSeaGreen3 = { 0.608, 0.804, 0.804 },
    DarkSeaGreen4 = { 0.41, 0.545, 0.545 },
    DarkSlateGray1 = { 0.592, 1, 1 },
    DarkSlateGray2 = { 0.552, 0.932, 0.932 },
    DarkSlateGray3 = { 0.475, 0.804, 0.804 },
    DarkSlateGray4 = { 0.32, 0.545, 0.545 },
    DeepPink1 = { 1, 0.08, 0.08 },
    DeepPink2 = { 0.932, 0.07, 0.07 },
    DeepPink3 = { 0.804, 0.064, 0.064 },
    DeepPink4 = { 0.545, 0.04, 0.04 },
    DeepSkyBlue1 = { 0, 0.75, 0.75 },
    DeepSkyBlue2 = { 0, 0.698, 0.698 },
    DeepSkyBlue3 = { 0, 0.604, 0.604 },
    DeepSkyBlue4 = { 0, 0.408, 0.408 },
    DodgerBlue1 = { 0.116, 0.565, 0.565 },
    DodgerBlue2 = { 0.11, 0.525, 0.525 },
    DodgerBlue3 = { 0.094, 0.455, 0.455 },
    DodgerBlue4 = { 0.064, 0.305, 0.305 },
    Firebrick1 = { 1, 0.19, 0.19 },
    Firebrick2 = { 0.932, 0.172, 0.172 },
    Firebrick3 = { 0.804, 0.15, 0.15 },
    Firebrick4 = { 0.545, 0.1, 0.1 },
    Gold1 = { 1, 0.844, 0.844 },
    Gold2 = { 0.932, 0.79, 0.79 },
    Gold3 = { 0.804, 0.68, 0.68 },
    Gold4 = { 0.545, 0.46, 0.46 },
    Goldenrod1 = { 1, 0.756, 0.756 },
```

```
    Goldenrod2 = { 0.932, 0.705, 0.705 },
    Goldenrod3 = { 0.804, 0.608, 0.608 },
    Goldenrod4 = { 0.545, 0.41, 0.41 },
    Green1 = { 0, 1, 1 },
    Green2 = { 0, 0.932, 0.932 },
    Green3 = { 0, 0.804, 0.804 },
    Green4 = { 0, 0.545, 0.545 },
    Honeydew1 = { 0.94, 1, 1 },
    Honeydew2 = { 0.88, 0.932, 0.932 },
    Honeydew3 = { 0.756, 0.804, 0.804 },
    Honeydew4 = { 0.512, 0.545, 0.545 },
    HotPink1 = { 1, 0.43, 0.43 },
    HotPink2 = { 0.932, 0.415, 0.415 },
    HotPink3 = { 0.804, 0.376, 0.376 },
    HotPink4 = { 0.545, 0.228, 0.228 },
    IndianRed1 = { 1, 0.415, 0.415 },
    IndianRed2 = { 0.932, 0.39, 0.39 },
    IndianRed3 = { 0.804, 0.332, 0.332 },
    IndianRed4 = { 0.545, 0.228, 0.228 },
    Ivory1 = { 1, 1, 1 },
    Ivory2 = { 0.932, 0.932, 0.932 },
    Ivory3 = { 0.804, 0.804, 0.804 },
    Ivory4 = { 0.545, 0.545, 0.545 },
    Khaki1 = { 1, 0.965, 0.965 },
    Khaki2 = { 0.932, 0.9, 0.9 },
    Khaki3 = { 0.804, 0.776, 0.776 },
    Khaki4 = { 0.545, 0.525, 0.525 },
    LavenderBlush1 = { 1, 0.94, 0.94 },
    LavenderBlush2 = { 0.932, 0.88, 0.88 },
    LavenderBlush3 = { 0.804, 0.756, 0.756 },
    LavenderBlush4 = { 0.545, 0.512, 0.512 },
    LemonChiffon1 = { 1, 0.98, 0.98 },
    LemonChiffon2 = { 0.932, 0.912, 0.912 },
    LemonChiffon3 = { 0.804, 0.79, 0.79 },
    LemonChiffon4 = { 0.545, 0.536, 0.536 },
    LightBlue1 = { 0.75, 0.936, 0.936 },
    LightBlue2 = { 0.698, 0.875, 0.875 },
    LightBlue3 = { 0.604, 0.752, 0.752 },
    LightBlue4 = { 0.408, 0.512, 0.512 },
    LightCyan1 = { 0.88, 1, 1 },
    LightCyan2 = { 0.82, 0.932, 0.932 },
    LightCyan3 = { 0.705, 0.804, 0.804 },
    LightCyan4 = { 0.48, 0.545, 0.545 },
    LightGoldenrod1 = { 1, 0.925, 0.925 },
    LightGoldenrod2 = { 0.932, 0.864, 0.864 },
    LightGoldenrod3 = { 0.804, 0.745, 0.745 },
    LightGoldenrod4 = { 0.545, 0.505, 0.505 },
    LightPink1 = { 1, 0.684, 0.684 },
    LightPink2 = { 0.932, 0.635, 0.635 },
    LightPink3 = { 0.804, 0.55, 0.55 },
    LightPink4 = { 0.545, 0.372, 0.372 },
    LightSalmon1 = { 1, 0.628, 0.628 },
    LightSalmon2 = { 0.932, 0.585, 0.585 },
    LightSalmon3 = { 0.804, 0.505, 0.505 },
    LightSalmon4 = { 0.545, 0.34, 0.34 },
    LightSkyBlue1 = { 0.69, 0.888, 0.888 },
    LightSkyBlue2 = { 0.644, 0.828, 0.828 },
    LightSkyBlue3 = { 0.552, 0.712, 0.712 },
    LightSkyBlue4 = { 0.376, 0.484, 0.484 },
    LightSteelBlue1 = { 0.792, 0.884, 0.884 },
    LightSteelBlue2 = { 0.736, 0.824, 0.824 },
    LightSteelBlue3 = { 0.635, 0.71, 0.71 },
    LightSteelBlue4 = { 0.43, 0.484, 0.484 },
    LightYellow1 = { 1, 1, 1 },
```

```
    LightYellow2 = { 0.932, 0.932, 0.932 },
    LightYellow3 = { 0.804, 0.804, 0.804 },
    LightYellow4 = { 0.545, 0.545, 0.545 },
    Magenta1 = { 1, 0, 0 },
    Magenta2 = { 0.932, 0, 0 },
    Magenta3 = { 0.804, 0, 0 },
    Magenta4 = { 0.545, 0, 0 },
    Maroon1 = { 1, 0.204, 0.204 },
    Maroon2 = { 0.932, 0.19, 0.19 },
    Maroon3 = { 0.804, 0.16, 0.16 },
    Maroon4 = { 0.545, 0.11, 0.11 },
    MediumOrchid1 = { 0.88, 0.4, 0.4 },
    MediumOrchid2 = { 0.82, 0.372, 0.372 },
    MediumOrchid3 = { 0.705, 0.32, 0.32 },
    MediumOrchid4 = { 0.48, 0.215, 0.215 },
    MediumPurple1 = { 0.67, 0.51, 0.51 },
    MediumPurple2 = { 0.624, 0.475, 0.475 },
    MediumPurple3 = { 0.536, 0.408, 0.408 },
    MediumPurple4 = { 0.365, 0.28, 0.28 },
    MistyRose1 = { 1, 0.894, 0.894 },
    MistyRose2 = { 0.932, 0.835, 0.835 },
    MistyRose3 = { 0.804, 0.716, 0.716 },
    MistyRose4 = { 0.545, 0.49, 0.49 },
    NavajoWhite1 = { 1, 0.87, 0.87 },
    NavajoWhite2 = { 0.932, 0.81, 0.81 },
    NavajoWhite3 = { 0.804, 0.7, 0.7 },
    NavajoWhite4 = { 0.545, 0.475, 0.475 },
    OliveDrab1 = { 0.752, 1, 1 },
    OliveDrab2 = { 0.7, 0.932, 0.932 },
    OliveDrab3 = { 0.604, 0.804, 0.804 },
    OliveDrab4 = { 0.41, 0.545, 0.545 },
    Orange1 = { 1, 0.648, 0.648 },
    Orange2 = { 0.932, 0.604, 0.604 },
    Orange3 = { 0.804, 0.52, 0.52 },
    Orange4 = { 0.545, 0.352, 0.352 },
    OrangeRed1 = { 1, 0.27, 0.27 },
    OrangeRed2 = { 0.932, 0.25, 0.25 },
    OrangeRed3 = { 0.804, 0.215, 0.215 },
    OrangeRed4 = { 0.545, 0.145, 0.145 },
    Orchid1 = { 1, 0.512, 0.512 },
    Orchid2 = { 0.932, 0.48, 0.48 },
    Orchid3 = { 0.804, 0.41, 0.41 },
    Orchid4 = { 0.545, 0.28, 0.28 },
    PaleGreen1 = { 0.604, 1, 1 },
    PaleGreen2 = { 0.565, 0.932, 0.932 },
    PaleGreen3 = { 0.488, 0.804, 0.804 },
    PaleGreen4 = { 0.33, 0.545, 0.545 },
    PaleTurquoise1 = { 0.732, 1, 1 },
    PaleTurquoise2 = { 0.684, 0.932, 0.932 },
    PaleTurquoise3 = { 0.59, 0.804, 0.804 },
    PaleTurquoise4 = { 0.4, 0.545, 0.545 },
    PaleVioletRed1 = { 1, 0.51, 0.51 },
    PaleVioletRed2 = { 0.932, 0.475, 0.475 },
    PaleVioletRed3 = { 0.804, 0.408, 0.408 },
    PaleVioletRed4 = { 0.545, 0.28, 0.28 },
    PeachPuff1 = { 1, 0.855, 0.855 },
    PeachPuff2 = { 0.932, 0.796, 0.796 },
    PeachPuff3 = { 0.804, 0.688, 0.688 },
    PeachPuff4 = { 0.545, 0.468, 0.468 },
    Pink1 = { 1, 0.71, 0.71 },
    Pink2 = { 0.932, 0.664, 0.664 },
    Pink3 = { 0.804, 0.57, 0.57 },
    Pink4 = { 0.545, 0.39, 0.39 },
    Plum1 = { 1, 0.732, 0.732 },
```

```
    Plum2 = { 0.932, 0.684, 0.684 },
    Plum3 = { 0.804, 0.59, 0.59 },
    Plum4 = { 0.545, 0.4, 0.4 },
    Purple1 = { 0.608, 0.19, 0.19 },
    Purple2 = { 0.57, 0.172, 0.172 },
    Purple3 = { 0.49, 0.15, 0.15 },
    Purple4 = { 0.332, 0.1, 0.1 },
    Red1 = { 1, 0, 0 },
    Red2 = { 0.932, 0, 0 },
    Red3 = { 0.804, 0, 0 },
    Red4 = { 0.545, 0, 0 },
    RosyBrown1 = { 1, 0.756, 0.756 },
    RosyBrown2 = { 0.932, 0.705, 0.705 },
    RosyBrown3 = { 0.804, 0.608, 0.608 },
    RosyBrown4 = { 0.545, 0.41, 0.41 },
    RoyalBlue1 = { 0.284, 0.464, 0.464 },
    RoyalBlue2 = { 0.264, 0.43, 0.43 },
    RoyalBlue3 = { 0.228, 0.372, 0.372 },
    RoyalBlue4 = { 0.152, 0.25, 0.25 },
    Salmon1 = { 1, 0.55, 0.55 },
    Salmon2 = { 0.932, 0.51, 0.51 },
    Salmon3 = { 0.804, 0.44, 0.44 },
    Salmon4 = { 0.545, 0.298, 0.298 },
    SeaGreen1 = { 0.33, 1, 1 },
    SeaGreen2 = { 0.305, 0.932, 0.932 },
    SeaGreen3 = { 0.264, 0.804, 0.804 },
    SeaGreen4 = { 0.18, 0.545, 0.545 },
    Seashell1 = { 1, 0.96, 0.96 },
    Seashell2 = { 0.932, 0.898, 0.898 },
    Seashell3 = { 0.804, 0.772, 0.772 },
    Seashell4 = { 0.545, 0.525, 0.525 },
    Sienna1 = { 1, 0.51, 0.51 },
    Sienna2 = { 0.932, 0.475, 0.475 },
    Sienna3 = { 0.804, 0.408, 0.408 },
    Sienna4 = { 0.545, 0.28, 0.28 },
    SkyBlue1 = { 0.53, 0.808, 0.808 },
    SkyBlue2 = { 0.494, 0.752, 0.752 },
    SkyBlue3 = { 0.424, 0.65, 0.65 },
    SkyBlue4 = { 0.29, 0.44, 0.44 },
    SlateBlue1 = { 0.512, 0.435, 0.435 },
    SlateBlue2 = { 0.48, 0.404, 0.404 },
    SlateBlue3 = { 0.41, 0.35, 0.35 },
    SlateBlue4 = { 0.28, 0.235, 0.235 },
    SlateGray1 = { 0.776, 0.888, 0.888 },
    SlateGray2 = { 0.725, 0.828, 0.828 },
    SlateGray3 = { 0.624, 0.712, 0.712 },
    SlateGray4 = { 0.424, 0.484, 0.484 },
    Snow1 = { 1, 0.98, 0.98 },
    Snow2 = { 0.932, 0.912, 0.912 },
    Snow3 = { 0.804, 0.79, 0.79 },
    Snow4 = { 0.545, 0.536, 0.536 },
    SpringGreen1 = { 0, 1, 1 },
    SpringGreen2 = { 0, 0.932, 0.932 },
    SpringGreen3 = { 0, 0.804, 0.804 },
    SpringGreen4 = { 0, 0.545, 0.545 },
    SteelBlue1 = { 0.39, 0.72, 0.72 },
    SteelBlue2 = { 0.36, 0.675, 0.675 },
    SteelBlue3 = { 0.31, 0.58, 0.58 },
    SteelBlue4 = { 0.21, 0.392, 0.392 },
    Tan1 = { 1, 0.648, 0.648 },
    Tan2 = { 0.932, 0.604, 0.604 },
    Tan3 = { 0.804, 0.52, 0.52 },
    Tan4 = { 0.545, 0.352, 0.352 },
    Thistle1 = { 1, 0.884, 0.884 },
```

```lua
    Thistle2 = { 0.932, 0.824, 0.824 },
    Thistle3 = { 0.804, 0.71, 0.71 },
    Thistle4 = { 0.545, 0.484, 0.484 },
    Tomato1 = { 1, 0.39, 0.39 },
    Tomato2 = { 0.932, 0.36, 0.36 },
    Tomato3 = { 0.804, 0.31, 0.31 },
    Tomato4 = { 0.545, 0.21, 0.21 },
    Turquoise1 = { 0, 0.96, 0.96 },
    Turquoise2 = { 0, 0.898, 0.898 },
    Turquoise3 = { 0, 0.772, 0.772 },
    Turquoise4 = { 0, 0.525, 0.525 },
    VioletRed1 = { 1, 0.244, 0.244 },
    VioletRed2 = { 0.932, 0.228, 0.228 },
    VioletRed3 = { 0.804, 0.196, 0.196 },
    VioletRed4 = { 0.545, 0.132, 0.132 },
    Wheat1 = { 1, 0.905, 0.905 },
    Wheat2 = { 0.932, 0.848, 0.848 },
    Wheat3 = { 0.804, 0.73, 0.73 },
    Wheat4 = { 0.545, 0.494, 0.494 },
    Yellow1 = { 1, 1, 1 },
    Yellow2 = { 0.932, 0.932, 0.932 },
    Yellow3 = { 0.804, 0.804, 0.804 },
    Yellow4 = { 0.545, 0.545, 0.545 },
    Gray0 = { 0.745, 0.745, 0.745 },
    Green0 = { 0, 1, 1 },
    Grey0 = { 0.745, 0.745, 0.745 },
    Maroon0 = { 0.69, 0.19, 0.19 },
    Purple0 = { 0.628, 0.125, 0.125 },
}

--- https://luarocks.org/modules/Firanel/lua-color
--- Copyright (c) 2021 Firanel

---https://github.com/Firanel/lua-color/blob/master/util/bitwise.lua
local bitwise = (function()
    -- Implementations of bitwise operators so that lua-color can be used
    -- with Lua 5.1 and LuaJIT 2.1.0-beta3 (e.g. inside Neovim).

    -- Code taken directly from:
    --
    ↪  https://stackoverflow.com/questions/5977654/how-do-i-use-the-bitwise-operator-xor-in-lua

    local function bit_xor(a, b)
      local p, c = 1, 0
      while a > 0 and b > 0 do
        local ra, rb = a % 2, b % 2
        if ra ~= rb then
          c = c + p
        end
        a, b, p = (a - ra) / 2, (b - rb) / 2, p * 2
      end
      if a < b then
        a = b
      end
      while a > 0 do
        local ra = a % 2
        if ra > 0 then
          c = c + p
        end
        a, p = (a - ra) / 2, p * 2
      end
      return c
    end
```

```lua
  local function bit_or(a, b)
    local p, c = 1, 0
    while a + b > 0 do
      local ra, rb = a % 2, b % 2
      if ra + rb > 0 then
        c = c + p
      end
      a, b, p = (a - ra) / 2, (b - rb) / 2, p * 2
    end
    return c
  end

  local function bit_not(n)
    local p, c = 1, 0
    while n > 0 do
      local r = n % 2
      if r < 1 then
        c = c + p
      end
      n, p = (n - r) / 2, p * 2
    end
    return c
  end

  local function bit_and(a, b)
    local p, c = 1, 0
    while a > 0 and b > 0 do
      local ra, rb = a % 2, b % 2
      if ra + rb > 1 then
        c = c + p
      end
      a, b, p = (a - ra) / 2, (b - rb) / 2, p * 2
    end
    return c
  end

  local function bit_lshift(x, by)
    return x * 2 ^ by
  end

  local function bit_rshift(x, by)
    return math.floor(x / 2 ^ by)
  end

  return {
    bit_xor = bit_xor,
    bit_or = bit_or,
    bit_not = bit_not,
    bit_and = bit_and,
    bit_lshift = bit_lshift,
    bit_rshift = bit_rshift,
  }
end)()

--- https://github.com/Firanel/lua-color/blob/master/util/class.lua
local class = (function()

  -- Code based on:
  -- http://lua-users.org/wiki/SimpleLuaClasses

  ---Helper function to create classes
  ---
  ---@usage local Color = class(function () --[[ constructor ]] end)
  ---@usage local Color2 = class(
```

```lua
  --   Color,
  --   function () --[[ constructor ]] end,
  --   { prop_a = "some value" }
  -- )
  local function class(base, init, defaults)
    local c = defaults or {} -- a new class instance
    if not init and type(base) == 'function' then
      init = base
      base = nil
    elseif type(base) == 'table' then
      -- our new class is a shallow copy of the base class!
      for i, v in pairs(base) do
        c[i] = v
      end
      c._base = base
    end
    -- the class will be the metatable for all its objects,
    -- and they will look up their methods in it.
    c.__index = c

    -- expose a constructor which can be called by <classname>(<args>)
    local mt = {}
    mt.__call = function(class_tbl, ...)
      local obj = {}
      setmetatable(obj, c)
      if init then
        init(obj, ...)
      else
        -- make sure that any stuff from the base class is initialized!
        if base and base.init then
          base.init(obj, ...)
        end
      end
      return obj
    end
    c.init = init
    c.is_a = function(self, klass)
      local m = getmetatable(self)
      while m do
        if m == klass then
          return true
        end
        m = m._base
      end
      return false
    end
    setmetatable(c, mt)
    return c
  end

  return class
end)()

---https://github.com/Firanel/lua-color/blob/master/utils/init.lua
local utils = (function()
  local function min_ind(first, ...)
    local min, ind = first, 1
    for i, v in ipairs { ... } do
      if v < min then
        min, ind = v, i + 1
      end
    end
    return min, ind
  end
```

```lua

  local function max_ind(first, ...)
    local max, ind = first, 1
    for i, v in ipairs { ... } do
      if v > max then
        max, ind = v, i + 1
      end
    end
    return max, ind
  end

  local function round(x)
    return x + 0.5 - (x + 0.5) % 1
  end

  local function clamp(x, min, max)
    return x < min and min or x > max and max or x
  end

  local function map(t, cb)
    local n = {}
    for i, v in ipairs(t) do
      n[i] = cb(v)
    end
    return n
  end

  return {
    min = min_ind,
    max = max_ind,
    round = round,
    clamp = clamp,
    map = map,
  }
end)()

---Source:
↪  [texmf-dist/tex/context/base/mkiv/attr-col.lua](https://git.texlive.info/texlive/tree/Master/texmf-dist/
local convert = (function()

  ---
  ---https://www.rapidtables.com/convert/color/rgb-to-cmyk.html
  ---https://www.101computing.net/cmyk-to-rgb-conversion-algorithm/
  ---
  ---@param r r # red (0.0 - 1.0)
  ---@param g g # green (0.0 - 1.0)
  ---@param b b # blue (0.0 - 1.0)
  ---
  ---@return c c # cyan (0.0 - 1.0)
  ---@return m m # magenta (0.0 - 1.0)
  ---@return y y # yellow (0.0 - 1.0)
  ---@return k k # key(black) (0.0 - 1.0)
  local function rgb_to_cmyk(r, g, b)
    local K = math.max(r, g, b)
    if K == 0 then
      return 0.0, 0.0, 0.0, 1.0
    end
    local k = 1 - K
    local c = (K - r) / K
    local m = (K - g) / K
    local y = (K - b) / K
    return c, m, y, k
  end

```

```lua
---https://www.rapidtables.com/convert/color/cmyk-to-rgb.html
local function cmyk_to_rgb(c, m, y, k)
  if not k then
    k = 0
  end
  ---texmf-dist/tex/context/base/mkiv/attr-col.lua
  -- local d = 1.0 - k
  -- local r = 1.0 - math.min(1.0, c * d + k)
  -- local g = 1.0 - math.min(1.0, m * d + k)
  -- local b = 1.0 - math.min(1.0, y * d + k)

  ---texmf-dist/tex/context/base/mkiv/attr-col.lua
  -- local r = 1.0 - math.min(1.0, c + k)
  -- local g = 1.0 - math.min(1.0, m + k)
  -- local b = 1.0 - math.min(1.0, y + k)


  ↪  ---https://github.com/Firanel/lua-color/blob/eba73e53e9abd2e8da4d56b016fd77b45c2f3b79/init.lua#L335-
  local K = 1 - k
  local r = (1 - c) * K
  local g = (1 - m) * K
  local b = (1 - y) * K

  return r, g, b

end

local function rgb_to_gray(r, g, b)
  if not r then
    return 0
  end
  local w = colors.weightgray
  if w == true then
    return .30 * r + .59 * g + .11 * b
  elseif not w then
    return r / 3 + g / 3 + b / 3
  else
    return w[1] * r + w[2] * g + w[3] * b
  end
end

local function cmyk_to_gray(c, m, y, k)
  return rgb_to_gray(cmyk_to_rgb(c, m, y, k))
end

-- http://en.wikipedia.org/wiki/HSI_color_space
-- http://nl.wikipedia.org/wiki/HSV_(kleurruimte)

--         h /= 60;        // sector 0 to 5
--         i = floor( h );
--         f = h - i;      // factorial part of h

local function hsv_to_rgb(h, s, v)
  if s > 1 then
    s = 1
  elseif s < 0 then
    s = 0
  elseif s == 0 then
    return v, v, v
  end
  if v > 1 then
    s = 1
  elseif v < 0 then
    v = 0
```

```
    end
    if h < 0 then
      h = 0
    elseif h >= 360 then
      h = mod(h, 360)
    end
    local hd = h / 60
    local hi = floor(hd)
    local f = hd - hi
    local p = v * (1 - s)
    local q = v * (1 - f * s)
    local t = v * (1 - (1 - f) * s)
    if hi == 0 then
      return v, t, p
    elseif hi == 1 then
      return q, v, p
    elseif hi == 2 then
      return p, v, t
    elseif hi == 3 then
      return p, q, v
    elseif hi == 4 then
      return t, p, v
    elseif hi == 5 then
      return v, p, q
    else
      print('error in hsv -> rgb', h, s, v)
      return 0, 0, 0
    end
  end

  local function rgb_to_hsv(r, g, b)
    local offset, maximum, other_1, other_2
    if r >= g and r >= b then
      offset, maximum, other_1, other_2 = 0, r, g, b
    elseif g >= r and g >= b then
      offset, maximum, other_1, other_2 = 2, g, b, r
    else
      offset, maximum, other_1, other_2 = 4, b, r, g
    end
    if maximum == 0 then
      return 0, 0, 0
    end
    local minimum = other_1 < other_2 and other_1 or other_2
    if maximum == minimum then
      return 0, 0, maximum
    end
    local delta = maximum - minimum
    return (offset + (other_1 - other_2) / delta) * 60, delta / maximum,
      maximum
  end

  local function gray_to_rgb(s) -- unweighted
    return 1 - s, 1 - s, 1 - s
  end

  local function hsv_to_gray(h, s, v)
    return rgb_to_gray(hsv_to_rgb(h, s, v))
  end

  local function gray_to_hsv(s)
    return 0, 0, s
  end

  ---
```

```
1402    ---@param h any # hue
1403    ---@param black any # black
1404    ---@param white any # white
1405    ---
1406    ---@return number r
1407    ---@return number g
1408    ---@return number b
1409    local function hwb_to_rgb(h, black, white)
1410      local r, g, b = hsv_to_rgb(h, 1, .5)
1411      local f = 1 - white - black
1412      return f * r + white, f * g + white, f * b + white
1413    end
1414
1415    return {
1416      rgb_to_cmyk = rgb_to_cmyk,
1417      cmyk_to_rgb = cmyk_to_rgb,
1418      rgb_to_gray = rgb_to_gray,
1419      cmyk_to_gray = cmyk_to_gray,
1420      hsv_to_rgb = hsv_to_rgb,
1421      rgb_to_hsv = rgb_to_hsv,
1422      gray_to_rgb = gray_to_rgb,
1423      hsv_to_gray = hsv_to_gray,
1424      gray_to_hsv = gray_to_hsv,
1425      hwb_to_rgb = hwb_to_rgb,
1426    }
1427
1428  end)()
1429
1430  --- https://github.com/Firanel/lua-color/blob/master/init.lua
1431
1432  ---The Class Color is the main class of the submodule. It represents a
1433  ---RGB color.
1434  ---
1435  ---@class Color
1436  ---@field r number # Red component.
1437  ---@field g number # Green component.
1438  ---@field b number # Blue component.
1439  ---@field a number # Alpha component.
1440  ---
1441  ---@function Color:__call
1442  ---
1443  ---@param value Color string|table|Color value (default: `nil`)
1444  ---
1445  ---@see Color:set
1446  local Color = (function()
1447
1448    ---Parse, convert and manipulate color values.
1449    ---
1450    -- @classmod Color
1451
1452    -- Lua 5.1 compat
1453    local bit_and = bitwise.bit_and
1454    local bit_lshift = bitwise.bit_lshift
1455    local bit_rshift = bitwise.bit_rshift
1456
1457    -- Utils
1458
1459    local function hcm_to_rgb(h, c, m)
1460      local r, g, b = 0, 0, 0
1461
1462      h = h * 6
1463      local x = c * (1 - math.abs(h % 2 - 1))
1464
1465      if h <= 1 then
```

```
      r, g, b = c, x, 0
    elseif h <= 2 then
      r, g, b = x, c, 0
    elseif h <= 3 then
      r, g, b = 0, c, x
    elseif h <= 4 then
      r, g, b = 0, x, c
    elseif h <= 5 then
      r, g, b = x, 0, c
    elseif h <= 6 then
      r, g, b = c, 0, x
    end

    return r + m, g + m, b + m
  end

  local function tonumPercent(str)
    if str:sub(-1) == '%' then
      return tonumber(str:sub(1, #str - 1)) / 100
    end
    return tonumber(str)
  end

  -- Color

  ---Color constructor.
  ---
  -- @function Color:__call
  ---
  ---@param ?string|table|Color value Color value (default: `nil`)
  ---
  ---@see Color:set

  ---Red component.
  -- @field r

  ---Green component.
  -- @field g

  ---Blue component.
  -- @field b

  ---Alpha component.
  -- @field a

  ---Color class
  local Color = class(nil, function(this, value)
    if value then
      if type(value) == 'string' then
        -- # gets expanded to ##
        value = string.gsub(value, '^##', '#')
      end
      this:set(value)
    end
  end, { __is_color = true, r = 0, g = 0, b = 0, a = 1 })

  ---Clone color
  ---
  ---@return Color copy
  function Color:clone()
    return Color(self)
  end

  ---Set color to value.
```

```
1530    -- <br>
1531    -- Called by constructor
1532    -- <br><br>
1533    -- Possible value types:
1534    -- <ul>
1535    --  <li>`Color`</li>
1536    --  <li>color name as specified in `Color.colorNames`</li>
1537    --  <li>css style functions as string:<ul>
1538    --   <li>`rgb(r, g, b)`</li>
1539    --   <li>`rgba(r, g, b, a)`</li>
1540    --   <li>`hsl(h, s, l)`</li>
1541    --   <li>`hsla(h, s, l, a)`</li>
1542    --   <li>`hsv(h, s, v)`</li>
1543    --   <li>`hsva(h, s, v, a)`</li>
1544    --   <li>`hwb(h, w, b)`</li>
1545    --   <li>`hwba(h, w, b, a)`</li>
1546    --   <li>`cmyk(c, m, y, k)`</li>
1547    --   </ul>
1548    --   Values are in the same ranges as in css ([0;255] for rgb, [0;1] for alpha,
      ↪  ...)<br>
1549    --   functions can be specified in a simplified syntax: `rgb(r, g, b) == rgb r g
      ↪  b`
1550    --  </li>
1551    --  <li>NCol string: `R10, 50%, 50%`</li>
1552    --  <li>hex string: `#rgb` | `#rgba` | `#rrggbb` | `#rrggbbaa` (`#` can be
      ↪  omitted)</li>
1553    --  <li>rgb values in [0;1]: `{r, g, b[, a]}` | `{r=r, g=g, b=b[, a=a]}`</li>
1554    --  <li>hsv values in [0;1]: `{h=h, s=s, v=v[, a=a]}`</li>
1555    --  <li>hsl values in [0;1]: `{h=h, s=s, l=l[, a=a]}`</li>
1556    --  <li>hwb values in [0;1]: `{h=h, w=w, b=b[, a=a]}`</li>
1557    --  <li>cmyk values in [0;1]: `{c=c, m=m, y=y, k=k}`</li>
1558    --  <li>single set mode, table with any combination of the following: <ul>
1559    --   <li>`red`</li>
1560    --   <li>`green`</li>
1561    --   <li>`blue`</li>
1562    --   <li>`alpha`</li>
1563    --   <li>`hue`</li>
1564    --   <li>`saturation`</li>
1565    --   <li>`value`</li>
1566    --   <li>`lightness`</li>
1567    --   <li>`whiteness`</li>
1568    --   <li>`blackness`</li>
1569    --   <li>`cyan`</li>
1570    --   <li>`magenta`</li>
1571    --   <li>`yellow`</li>
1572    --   <li>`key`</li>
1573    --   </ul>
1574    --   All values are in `[0;1]`.<br>
1575    --   They will be applied in the order: `rgba -> hsl -> hwb -> hsv -> cmyk`<br>
1576    --   If `lightness` is given, saturation is treated as hsl saturation,
1577    --   otherwise it will be treated as hsv saturation.
1578    --  </li>
1579    -- </ul>
1580    ---
1581    ---@see Color:__call
1582    ---
1583    ---@param value string|table|Color
1584    ---
1585    ---@return Color self
1586    ---
1587    ---@usage color:set "#f1f1f1"
1588    ---@usage color:set "rgba(241, 241, 241, 0.5)"
1589    ---@usage color:set "hsl 180 100% 20%"
1590    ---@usage color:set { r = 0.255, g = 0.729, b = 0.412 }
```

```
1591    ---@usage color:set { 0.255, 0.729, 0.412 } -- same as above
1592    ---@usage color:set { h = 0.389, s = 0.65, v = 0.73 }
1593    function Color:set(value)
1594      assert(value)
1595
1596      -- from Color
1597      if value.__is_color then
1598        self.r = value.r
1599        self.g = value.g
1600        self.b = value.b
1601        self.a = value.a
1602
1603      elseif type(value) == 'string' then
1604        self.a = 1
1605
1606        if value:sub(1, 1) ~= '#' then
1607          local c = colors[value]
1608          if c then
1609            self.r = c[1]
1610            self.g = c[2]
1611            self.b = c[3]
1612            return
1613          end
1614
1615          local func, values = value:match '(%w+)[ %(]+([x ,.%x%%]+)'
1616          if func ~= nil then
1617            if func == 'rgb' then
1618              local r, g, b =
1619                values:match '([x.%x]+)[ ,]+([x.%x]+)[ ,]+([x.%x]+)'
1620              assert(r and g and b)
1621              self.r = tonumber(r) / 0xff
1622              self.g = tonumber(g) / 0xff
1623              self.b = tonumber(b) / 0xff
1624              return self
1625            elseif func == 'rgba' then
1626              local r, g, b, a =
1627                values:match '([x.%x]+)[ ,]+([x.%x]+)[ ,]+([x.%x]+)[ ,]+([x.%x]+%%?)'
1628              assert(r and g and b and a)
1629              self.r = tonumber(r) / 0xff
1630              self.g = tonumber(g) / 0xff
1631              self.b = tonumber(b) / 0xff
1632              self.a = tonumPercent(a)
1633              return self
1634            elseif func == 'hsv' then
1635              local h, s, v =
1636                values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
1637              assert(h and s and v)
1638              return self:set{
1639                h = tonumber(h) / 360,
1640                s = tonumPercent(s),
1641                v = tonumPercent(v),
1642              }
1643            elseif func == 'hsva' then
1644              local h, s, v, a =
1645                values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[
                    ↪  ,]+([x.%x]+%%?)'
1646              assert(h and s and v and a)
1647              return self:set{
1648                h = tonumber(h) / 360,
1649                s = tonumPercent(s),
1650                v = tonumPercent(v),
1651                a = tonumPercent(a),
1652              }
1653            elseif func == 'hsl' then
```

```
1654              local h, s, l =
1655                values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
1656              assert(h and s and l)
1657              return self:set{
1658                h = tonumber(h) / 360,
1659                s = tonumPercent(s),
1660                l = tonumPercent(l),
1661              }
1662            elseif func == 'hsla' then
1663              local h, s, l, a =
1664                values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[
                  ↪  ,]+([x.%x]+%%?)'
1665              assert(h and s and l and a)
1666              return self:set{
1667                h = tonumber(h) / 360,
1668                s = tonumPercent(s),
1669                l = tonumPercent(l),
1670                a = tonumPercent(a),
1671              }
1672            elseif func == 'hwb' then
1673              local h, w, b =
1674                values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
1675              assert(h and w and b)
1676              return self:set{
1677                h = tonumber(h) / 360,
1678                w = tonumPercent(w),
1679                b = tonumPercent(b),
1680              }
1681            elseif func == 'hwba' then
1682              local h, w, b, a =
1683                values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[
                  ↪  ,]+([x.%x]+%%?)'
1684              assert(h and w and b and a)
1685              return self:set{
1686                h = tonumber(h) / 360,
1687                w = tonumPercent(w),
1688                b = tonumPercent(b),
1689                a = tonumPercent(a),
1690              }
1691            elseif func == 'cmyk' then
1692              local c, m, y, k =
1693                values:match '([x.%x]+%%?)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[
                  ↪  ,]+([x.%x]+%%?)'
1694              assert(c and m and y and k)
1695              return self:set{
1696                c = tonumPercent(c),
1697                m = tonumPercent(m),
1698                y = tonumPercent(y),
1699                k = tonumPercent(k),
1700              }
1701            end
1702          else
1703            local col, dist, w, b, a =
1704              value:match '([RGBCMYrgbcmy])(%d*)[, ]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[
                  ↪  ,]+([x.%x]+%%?)'
1705            if col == nil then
1706              col, dist, w, b, a =
1707                value:match '([RGBCMYrgbcmy])(%d*)[, ]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
1708            end
1709            if col then
1710              col = col:lower()
1711
1712              local h
1713              if col == 'r' then
```

```
              h = 0
            elseif col == 'y' then
              h = 1 / 6
            elseif col == 'g' then
              h = 2 / 6
            elseif col == 'c' then
              h = 3 / 6
            elseif col == 'b' then
              h = 4 / 6
            elseif col == 'm' then
              h = 5 / 6
            end

            if #dist > 0 then
              h = h + tonumber(dist) / 600
            end

            return self:set{
              h = h,
              w = tonumPercent(w),
              b = tonumPercent(b),
              a = a and tonumPercent(a) or 1,
            }
          end
        end
      else
        value = value:sub(2)
      end

      local pattern
      local div = 0xff
      if #value == 3 then
        pattern = '(%x)(%x)(%x)'
        div = 0xf
      elseif #value == 4 then
        pattern = '(%x)(%x)(%x)(%x)'
        div = 0xf
      elseif #value == 6 then
        pattern = '(%x%x)(%x%x)(%x%x)'
      elseif #value == 8 then
        pattern = '(%x%x)(%x%x)(%x%x)(%x%x)'
      else
        error('Not a valid color: ' .. tostring(value))
      end
      local r, g, b, a = value:match(pattern)
      assert(r ~= nil, 'Not a valid color: ' .. tostring(value))
      self.r = tonumber(r, 16) / div
      self.g = tonumber(g, 16) / div
      self.b = tonumber(b, 16) / div
      self.a = a ~= nil and tonumber(a, 16) / div.a or 1

      -- table with rgb
    elseif value[1] ~= nil then
      self.r = value[1]
      self.g = value[2]
      self.b = value[3]
      self.a = value[4] or self.a or 1
    elseif value.r ~= nil then
      self.r = value.r
      self.g = value.g or self.g
      self.b = value.b or self.b
      self.a = value.a or self.a

    elseif value.c ~= nil then
```

```
      self.r, self.g, self.b = convert.cmyk_to_rgb(value.c, value.m,
        value.y, value.k)
      self.a = 1

      -- table with hs[vl]
    elseif value.h ~= nil then
      if value.w ~= nil then -- hwb
        value.v = 1 - value.b
        value.s = 1 - value.w / value.v
      end

      local hue, saturation = value.h, value.s
      assert(hue ~= nil, saturation ~= nil)

      local r, g, b = 0, 0, 0

      if value.v ~= nil then
        local v = value.v
        local chroma = saturation * v
        r, g, b = hcm_to_rgb(hue, chroma, v - chroma)

      elseif value.l ~= nil then
        local lightness = value.l
        local chroma = (1 - math.abs(2 * lightness - 1)) * saturation
        r, g, b = hcm_to_rgb(hue, chroma, lightness - chroma / 2)
      end

      self.r = r
      self.g = g
      self.b = b
      self.a = value.a or self.a or 1

    else -- Single set mode
      if value.red then
        self.r = value.red
      end
      if value.green then
        self.g = value.green
      end
      if value.blue then
        self.b = value.blue
      end
      if value.alpha then
        self.a = value.alpha
      end

      if value.lightness then
        local h, s, l = self:hsl()
        self:set{
          h = value.hue or h,
          s = value.saturation or s,
          l = value.lightness or l,
        }
        value.hue = nil
        value.saturation = nil
      end

      if value.whiteness or value.blackness then
        local h, w, b = self:hwb()
        self:set{
          h = value.hue or h,
          w = value.whiteness or w,
          b = value.backness or b,
        }
```

```
1842          value.hue = nil
1843        end
1844
1845        if value.hue or value.saturation or value.value then
1846          local h, s, v = self:hsv()
1847          self:set{
1848            h = value.hue or h,
1849            s = value.saturation or s,
1850            v = value.value or v,
1851          }
1852        end
1853
1854        if value.cyan or value.magenta or value.yellow or value.key then
1855          local c, m, y, k = self:cmyk()
1856          self:set{
1857            c = value.cyan or c,
1858            m = value.magenta or m,
1859            y = value.yellow or y,
1860            k = value.key or k,
1861          }
1862        end
1863      end
1864
1865      local r, g, b, a = utils.clamp(self.r, 0, 1),
1866        utils.clamp(self.g, 0, 1), utils.clamp(self.b, 0, 1),
1867        utils.clamp(self.a, 0, 1)
1868      assert(r and g and b and a, 'Color invalid')
1869      return self
1870    end
1871
1872    ---Get rgb values.
1873    ---
1874    ---@return number[0;1] red
1875    ---@return number[0;1] green
1876    ---@return number[0;1] blue
1877    function Color:rgb()
1878      return self.r, self.g, self.b
1879    end
1880
1881    ---Get rgba values.
1882    ---
1883    ---@return number[0;1] red
1884    ---@return number[0;1] green
1885    ---@return number[0;1] blue
1886    ---@return number[0;1] alpha
1887    function Color:rgba()
1888      return self.r, self.g, self.b, self.a
1889    end
1890
1891    function Color:_hsvm()
1892      local r, g, b = self.r, self.g, self.b
1893
1894      local max, max_i = utils.max(r, g, b)
1895      local min = math.min(r, g, b)
1896      local chroma = max - min
1897
1898      local hue
1899      if chroma == 0 then
1900        hue = 0
1901      elseif max_i == 1 then
1902        hue = ((g - b) / chroma) / 6
1903      elseif max_i == 2 then
1904        hue = (2 + (b - r) / chroma) / 6
1905      elseif max_i == 3 then
```

```
      hue = (4 + (r - g) / chroma) / 6
    end

    local saturation = max == 0 and 0 or chroma / max

    return hue, saturation, max, min
  end

  ---Get hsv values.
  ---
  ---@return number[0;1] hue
  ---@return number[0;1] saturation
  ---@return number[0;1] value
  function Color:hsv()
    local h, s, v = self:_hsvm()
    return h, s, v
  end

  ---Get hsv values.
  ---
  ---@return number[0;1] hue
  ---@return number[0;1] saturation
  ---@return number[0;1] value
  ---@return number[0;1] alpha
  function Color:hsva()
    local h, s, v = self:_hsvm()
    return h, s, v, self.a
  end

  ---Get hsl values.
  ---
  ---@return number[0;1] hue
  ---@return number[0;1] saturation
  ---@return number[0;1] lightness
  function Color:hsl()
    local hue, _, max, min = self:_hsvm()
    local lightness = (max + min) / 2

    local saturation = lightness == 0 and 0 or (max - lightness) /
                         math.min(lightness, 1 - lightness)

    if saturation ~= saturation then
      saturation = 0
    end

    return hue, saturation, lightness
  end

  ---Get hsl values.
  ---
  ---@return number[0;1] hue
  ---@return number[0;1] saturation
  ---@return number[0;1] lightness
  ---@return number[0;1] alpha
  function Color:hsla()
    local h, s, l = self:hsl()
    return h, s, l, self.a
  end

  ---Get hwb values.
  ---
  ---@return number[0;1] hue
  ---@return number[0;1] whiteness
  ---@return number[0;1] blackness
```

```lua
function Color:hwb()
  local h, s, v = self:hsv()
  local w = (1 - s) * v
  local b = 1 - v
  return h, w, b
end

---Get hwb values.
---
---@return number[0;1] hue
---@return number[0;1] whiteness
---@return number[0;1] blackness
---@return number[0;1] alpha
function Color:hwba()
  local h, w, b = self:hwb()
  return h, w, b, self.a
end

---Get cmyk values.
---
---@return number[0;1] cyan
---@return number[0;1] magenta
---@return number[0;1] yellow
---@return number[0;1] key
function Color:cmyk()
  return convert.rgb_to_cmyk(self.r, self.g, self.b)
end

---Rotate hue of color.
---
---@param number[0;1]|table value Part of full turn or table containing degree or radians
---
---@return Color self
---
---@usage color:rotate(0.5)
---@usage color:rotate {deg=180}
---@usage color:rotate {rad=math.pi}
function Color:rotate(value)
  local r
  if type(value) == 'number' then
    r = value
  elseif value.rad ~= nil then
    r = value.rad / (math.pi * 2)
  elseif value.deg ~= nil then
    r = value.deg / 360
  else
    error('No valid argument')
  end

  local h, s, v = self:hsv()
  h = (h + r) % 1
  self:set{ h = h, s = s, v = v, a = self.a }

  return self
end

---Invert the color.
---
---@return Color self
function Color:invert()
  self.r = 1 - self.r
  self.g = 1 - self.g
  self.b = 1 - self.b
```

```
2033      return self
2034    end
2035
2036    ---Reduce saturation to 0.
2037    ---
2038    ---@return Color self
2039    function Color:grey()
2040      local h, _, v = self:hsv()
2041      self:set{ h = h, s = 0, v = v, a = self.a }
2042      return self
2043    end
2044
2045    ---Set to black or white depending on lightness.
2046    ---
2047    ---@param ?number[0;1] lightness Cutoff point (Default: 0.5)
2048    ---
2049    ---@return Color self
2050    function Color:blackOrWhite(lightness)
2051      local _, _, l = self:hsl()
2052      local v = l > lightness and 1 or 0
2053      self.r = v
2054      self.g = v
2055      self.b = v
2056      return self
2057    end
2058
2059    ---Mix two colors together.
2060    ---
2061    ---@param Color other
2062    ---@param ?number strength 0 results in self, 1 results in other (Default: 0.5)
2063    ---
2064    ---@return Color self
2065    function Color:mix(other, strength)
2066      if strength == nil then
2067        strength = 0.5
2068      end
2069      self.r = self.r * (1 - strength) + other.r * strength
2070      self.g = self.g * (1 - strength) + other.g * strength
2071      self.b = self.b * (1 - strength) + other.b * strength
2072      self.a = self.a * (1 - strength) + other.a * strength
2073      return self
2074    end
2075
2076    ---Generate complementary color.
2077    ---
2078    ---@return Color
2079    function Color:complement()
2080      return Color(self):rotate(0.5)
2081    end
2082
2083    ---Generate analogous color scheme.
2084    ---
2085    ---@return Color
2086    ---@return Color self
2087    ---@return Color
2088    function Color:analogous()
2089      local h, s, v = self:hsv()
2090      return Color { h = (h - 1 / 12) % 1, s = s, v = v, a = self.a },
2091        self, Color { h = (h + 1 / 12) % 1, s = s, v = v, a = self.a }
2092    end
2093
2094    ---Generate triadic color scheme.
2095    ---
2096    ---@return Color self
```

```lua
---@return Color
---@return Color
function Color:triad()
  local h, s, v = self:hsv()
  return self,
    Color { h = (h + 1 / 3) % 1, s = s, v = v, a = self.a },
    Color { h = (h + 2 / 3) % 1, s = s, v = v, a = self.a }
end

---Generate tetradic color scheme.
---
---@return Color self
---@return Color
---@return Color
---@return Color
function Color:tetrad()
  local h, s, v = self:hsv()
  return self,
    Color { h = (h + 1 / 4) % 1, s = s, v = v, a = self.a },
    Color { h = (h + 2 / 4) % 1, s = s, v = v, a = self.a },
    Color { h = (h + 3 / 4) % 1, s = s, v = v, a = self.a }
end

---Generate compound color scheme.
---
---@return Color
---@return Color self
---@return Color
function Color:compound()
  local ca, _, cb = self:complement():analogous()
  return ca, self, cb
end

---Generate evenly spaced color scheme.
-- <br>
-- Generalization of `triad` and `tetrad`.
---
---@param int     n Return n colors
---@param ?number r Space colors over r rotations (Default: 1)
---
---@return {Color,...} Table with n colors including self at index 1
function Color:evenlySpaced(n, r)
  assert(n > 0, 'n needs to be greater than 0')
  r = r or 1

  local res = { self }

  local rot = r / n
  local h, s, v = self:hsv()
  local a = self.a

  for i = 1, n - 1 do
    h = (h + rot) % 1
    table.insert(res, Color { h = h, s = s, v = v, a = a })
  end

  return res
end

---Get string representation of color.
---
-- If `format` is `nil`, `color:tostring()` is the same as `tostring(color)`.
---
---@param ?string format One of: `#fff`, `#ffff`, `#ffffff`, `#ffffffff`,
```

```lua
--  rgb, rgba, hsv, hsva, hsl, hsla, hwb, hwba, ncol, cmyk
---
---@return string
---
---@see Color:__tostring
function Color:tostring(format)
  if format == nil then
    return tostring(self)
  end

  format = format:lower()

  if format:sub(1, 1) == '#' then
    if #format == 4 then
      return string.format('#%x%x%x', utils.round(self.r * 0xf),
        utils.round(self.g * 0xf), utils.round(self.b * 0xf))
    elseif #format == 5 then
      return string.format('#%x%x%x%x', utils.round(self.r * 0xf),
        utils.round(self.g * 0xf), utils.round(self.b * 0xf),
        utils.round(self.a * 0xf))
    elseif #format == 7 then
      return string.format('#%02x%02x%02x',
        utils.round(self.r * 0xff), utils.round(self.g * 0xff),
        utils.round(self.b * 0xff))
    elseif #format == 9 then
      return string.format('#%02x%02x%02x%02x',
        utils.round(self.r * 0xff), utils.round(self.g * 0xff),
        utils.round(self.b * 0xff), utils.round(self.a * 0xff))
    end
  elseif format == 'rgb' then
    return string.format('rgb(%d, %d, %d)',
      utils.round(self.r * 0xff), utils.round(self.g * 0xff),
      utils.round(self.b * 0xff))
  elseif format == 'rgba' then
    return string.format('rgba(%d, %d, %d, %s)',
      utils.round(self.r * 0xff), utils.round(self.g * 0xff),
      utils.round(self.b * 0xff), self.a)
  elseif format == 'hsv' then
    local h, s, v = self:hsv()
    return string.format('hsv(%d, %d%%, %d%%)', utils.round(h * 360),
      utils.round(s * 100), utils.round(v * 100))
  elseif format == 'hsva' then
    local h, s, v, a = self:hsva()
    return string.format('hsva(%d, %d%%, %d%%, %s)',
      utils.round(h * 360), utils.round(s * 100),
      utils.round(v * 100), a)
  elseif format == 'hsl' then
    local h, s, l = self:hsl()
    return string.format('hsl(%d, %d%%, %d%%)', utils.round(h * 360),
      utils.round(s * 100), utils.round(l * 100))
  elseif format == 'hsla' then
    local h, s, l, a = self:hsla()
    return string.format('hsla(%d, %d%%, %d%%, %s)',
      utils.round(h * 360), utils.round(s * 100),
      utils.round(l * 100), a)
  elseif format == 'hwb' then
    local h, w, b = self:hwb()
    return string.format('hwb(%d, %d%%, %d%%)', utils.round(h * 360),
      utils.round(w * 100), utils.round(b * 100))
  elseif format == 'hwba' then
    local h, w, b, a = self:hwba()
    return string.format('hwba(%d, %d%%, %d%%, %s)',
      utils.round(h * 360), utils.round(w * 100),
      utils.round(b * 100), a)
```

```
2225      elseif format == 'ncol' then
2226        local h, w, b = self:hwb()
2227        local h_maj, h_min = math.modf(h * 6)
2228        h_maj = h_maj % 6
2229
2230        local col
2231        if h_maj == 0 then
2232          col = 'R'
2233        elseif h_maj == 1 then
2234          col = 'Y'
2235        elseif h_maj == 2 then
2236          col = 'G'
2237        elseif h_maj == 3 then
2238          col = 'C'
2239        elseif h_maj == 4 then
2240          col = 'B'
2241        else
2242          col = 'M'
2243        end
2244
2245        return string.format('%s%d, %d%%, %d%%', col,
2246          utils.round(h_min * 100), utils.round(w * 100),
2247          utils.round(b * 100))
2248      elseif format == 'cmyk' then
2249        local c, m, y, k = self:cmyk()
2250        return string.format('cymk(%d%%, %d%%, %d%%, %d%%)',
2251          utils.round(c * 100), utils.round(m * 100),
2252          utils.round(y * 100), utils.round(k * 100))
2253      end
2254
2255      return tostring(self)
2256    end
2257
2258    ---Get color in rgb hex notation.
2259    -- <br>
2260    -- only adds alpha value if `color.a < 1`
2261    ---
2262    ---@return string `#rrggbb` | `#rrggbbaa`
2263    ---
2264    ---@see Color:tostring
2265    function Color:__tostring()
2266      if self.a < 1 then
2267        return string.format('#%02x%02x%02x%02x',
2268          utils.round(self.r * 0xff), utils.round(self.g * 0xff),
2269          utils.round(self.b * 0xff), utils.round(self.a * 0xff))
2270      else
2271        return string.format('#%02x%02x%02x', utils.round(self.r * 0xff),
2272          utils.round(self.g * 0xff), utils.round(self.b * 0xff))
2273      end
2274    end
2275
2276    ---Check if colors are equal.
2277    ---
2278    ---@param Color other
2279    ---
2280    ---@return boolean all values are equal
2281    function Color:__eq(other)
2282      return
2283        self.r == other.r and self.g == other.g and self.b == other.b and
2284          self.a == other.a
2285    end
2286
2287    ---Checks whether color is darker.
2288    ---
```

```lua
---@param Color other
---
---@return boolean self is darker than other
function Color:__lt(other)
  local _, _, la = self:hsl()
  local _, _, lb = other:hsl()
  return la < lb
end

---Checks whether color is as dark or darker.
---
---@param Color other
---
---@return boolean self is as dark or darker than other
function Color:__le(other)
  local _, _, la = self:hsl()
  local _, _, lb = other:hsl()
  return la <= lb
end

---Iterate through color.
---
-- Iterates through r, g, b, and a.
function Color:__pairs()
  local function iter(tbl, k)
    if k == nil then
      return 'r', self.r
    elseif k == 'r' then
      return 'g', self.g
    elseif k == 'g' then
      return 'b', self.b
    elseif k == 'b' then
      return 'a', self.a
    end
  end

  return iter, self, nil
end

---Get inverted clone of color.
---
---@return Color
function Color:__unm()
  return Color(self):invert()
end

---Mix two colors evenly.
---
---@param Color a first color
---@param Color b second color
---
---@return Color new color
---
---@see Color:mix
function Color.__add(a, b)
  assert(Color.isColor(a) and Color.isColor(b),
    'Can only add two colors.')
  return Color(a):mix(b)
end

---Complement of even mix.
---
---@param Color a first color
---@param Color b second color
```

```
2353    ---
2354    ---@return Color new color
2355    ---
2356    ---@see Color:mix
2357    ---@see Color.__add
2358    function Color.__sub(a, b)
2359      assert(Color.isColor(a) and Color.isColor(b),
2360        'Can only add two colors.')
2361      return Color(a):mix(b):rotate(0.5)
2362    end
2363
2364    ---Apply rgb mask to color.
2365    ---
2366    ---@param Color|number a color or mask
2367    ---@param Color|number b color or mask (if a and b are colors b is used as mask)
2368    ---
2369    ---@return Color new color
2370    ---
2371    ---@usage local new_col = color & 0xff00ff -- get new color without the green
       ↪  channel
2372    function Color.__band(a, b)
2373      local color, mask
2374      if Color.isColor(a) and type(b) == 'number' then
2375        color = a
2376        mask = b
2377      elseif Color.isColor(b) and type(a) == 'number' then
2378        color = b
2379        mask = a
2380      elseif Color.isColor(a) and Color.isColor(b) then
2381        color = a
2382        mask = bit_lshift(utils.round(b.r * 0xff), 16) +
2383                 bit_lshift(utils.round(b.g * 0xff), 8) +
2384                 utils.round(b.b * 0xff)
2385      else
2386        error(
2387          'Required arguments: Color|number,Color|number Received: ' ..
2388            type(a) .. ',' .. type(b))
2389      end
2390
2391      return Color {
2392        bit_and(utils.round(color.r * 0xff), bit_rshift(mask, 16)) / 0xff,
2393        bit_and(utils.round(color.g * 0xff), bit_rshift(mask, 8)) / 0xff,
2394        bit_and(utils.round(color.b * 0xff), mask) / 0xff,
2395        color.a,
2396      }
2397    end
2398
2399    ---Apply rgb mask to color, providing backwards compatibility for Lua 5.1 and
       ↪  LuaJIT 2.1.0-beta3 (e.g. inside Neovim), which don't provide native support
       ↪  for bitwise operators.
2400    ---
2401    ---@param a Color|number # color or mask
2402    ---@param b Color|number # color or mask (if a and b are colors b is used as mask)
2403    ---
2404    ---@return Color new color
2405    ---
2406    ---@usage local new_col = Color.band(color, 0xff00ff) -- get new color without the
       ↪  green channel
2407    function Color.band(a, b)
2408      return Color.__band(a, b)
2409    end
2410
2411    ---Check whether `color` is a Color.
2412    ---
```

```
2413    ---@param color Color
2414    ---
2415    ---@return boolean # is a color
2416    ---
2417    ---@usage if Color.isColor(color) then print "It's a color!" end
2418    function Color.isColor(color)
2419      return color ~= nil and color.__is_color == true
2420    end
2421
2422    ---Format a PDF colorstack string. This string can be assigned to the
2423    ---`node.data` field of a PDF colorstock node.
2424    ---
2425    ---@return string # A string like this example `1 0 0 rg 1 0 0 RG`
2426    function Color:format_pdf_colorstack_string()
2427      return table.concat({
2428        self.r,
2429        self.g,
2430        self.b,
2431        'rg',
2432        self.r,
2433        self.g,
2434        self.b,
2435        'RG',
2436      }, ' ')
2437    end
2438
2439    ---Create a PDF colorstack node.
2440    ---
2441    ---@param command "set"|"push"|"pop"|"current"
2442    ---
2443    ---@return PdfColorstackWhatsitNode
2444    function Color:create_pdf_colorstack_node(command)
2445      local whatsit = node.new('whatsit', 'pdf_colorstack') --[[@as
       ↪  PdfColorstackWhatsitNode]]
2446      if command == 'set' then
2447        whatsit.command = 0
2448      elseif command == 'push' then
2449        whatsit.command = 1
2450      elseif command == 'pop' then
2451        whatsit.command = 2
2452      elseif command == 'current' then
2453        whatsit.command = 3
2454      end
2455      if command ~= 'pop' then
2456        whatsit.data = self:format_pdf_colorstack_string()
2457      end
2458      return whatsit
2459    end
2460
2461    ---Write a PDF colorstock node using `node.write()`.
2462    ---
2463    ---@param command "set"|"push"|"pop"|"current"
2464    ---
2465    function Color:write_pdf_colorstack_node(command)
2466      node.write(self:create_pdf_colorstack_node(command))
2467    end
2468
2469    function Color:write_box()
2470      self:write_pdf_colorstack_node('push')
2471      local rule = node.new('rule') --[[@as RuleNode]]
2472      rule.width = tex.sp('0.5cm')
2473      rule.height = tex.sp('0.5cm')
2474      node.write(rule)
2475      self:write_pdf_colorstack_node('pop')
```

```lua
  end

  return Color

end)()

---
---@param scheme 'base'|'svg'|'x11'
local function print_color_table(scheme)
  for _, name in pairs(schemes[scheme]) do
    -- local color = Color({ r = rgb[1], g = rgb[2], b = rgb[3] })
    local color = colors[name]
    local r = utils.round(color[1] * 255)
    local g = utils.round(color[2] * 255)
    local b = utils.round(color[3] * 255)
    tex.print('\\par\\noindent')
    tex.print(string.format('\\FarbeBox{rgba(%d, %d, %d, 1}', r, g, b))
    tex.print('\\quad{}' .. name)
  end
end

---
---@param operator string # The PDF color operator, e. g. `0.2 0.5 1 rg 0.2 0.5 1 RG`
---
---@return Color|nil
local function convert_pdf_color_operator(operator)
  operator = operator:lower()

  ---
  ---@param count_floats integer
  ---@param suffix string
  ---
  ---@return string
  local function build_pattern(count_floats, suffix)
    local pattern_float = '(%d*%.?%d+)'

    local patterns = {}
    for i = 1, count_floats, 1 do
      patterns[i] = pattern_float
    end
    patterns[count_floats + 1] = suffix

    return table.concat(patterns, ' +')
  end

  local n = tonumber

  local r, g, b = operator:match(build_pattern(3, 'rg'))
  if r ~= nil then
    log.debug('operator to RGB: %s %s %s', r, g, b)
    return Color({ r = n(r), g = n(g), b = n(b) })
  end

  local c, m, y, k = operator:match(build_pattern(3, 'k'))
  if c ~= nil then
    log.debug('operator to CMYK: %s %s %s %s', c, m, y, k)
    return Color({ c = n(c), m = n(m), y = n(y), k = n(k) })
  end

  local gray = operator:match(build_pattern(1, 'g'))
  if gray ~= nil then
    log.debug('operator to GRAY: %s', gray)
    return Color({ r = n(gray), g = n(gray), b = n(gray) })
```

```
2539    end
2540  end
2541
2542  ---
2543  ---@param name string # The name of the color.
2544  ---@param operator string # The PDF color operator, e. g. `0.2 0.5 1 rg 0.2 0.5 1
      ↪  RG`
2545  local function import_color(name, operator)
2546    if colors[name] ~= nil then
2547      return
2548    end
2549    local color = convert_pdf_color_operator(operator)
2550    log.info('Import new color: name %s, operator: %s, converted: %s',
2551      name, operator, color)
2552    if color ~= nil then
2553      colors[name] = { color.r, color.g, color.b }
2554    end
2555  end
2556
2557  return {
2558    convert = convert,
2559    Color = Color --[[@as Color]] ,
2560    print_color_table = print_color_table,
2561    import_color = import_color,
2562  }
```

4.2 farbe.tex

```
%% farbe.tex
%% Copyright 2025 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files farbe.lua, farbe.tex,
% and farbe.sty.

\directlua
{
  if farbe == nil then
    farbe = require('farbe')
  end
}

\def\FarbePdfLiteral#1{\csname\string\color@#1\endcsname}

\def\FarbeImport#1{%
  \directlua
  {
    farbe.import_color('#1', '\FarbePdfLiteral{#1}')
  }%
}

\def\FarbeColor#1
{%
  \directlua{farbe.Color('#1'):write_pdf_colorstack_node('push')}%
}

\def\FarbeTextColor#1#2
{%
  \protect\leavevmode{%
  \directlua{farbe.Color('#1'):write_pdf_colorstack_node('push')}%
  #2%
  \directlua{farbe.Color('#1'):write_pdf_colorstack_node('pop')}%
  }
}

\def\FarbeBox#1
{%
  \directlua{farbe.Color('#1'):write_box()}%
}
```

4.3 farbe.sty

```
%% farbe.sty
%% Copyright 2025 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files farbe.lua, farbe.tex,
% and farbe.sty.

\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{farbe}[2025/05/31 v0.1.0 Color management (conversion, names) for
↪  LuaTeX implemented in Lua.]

\input farbe.tex
```