The farbe package

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

0.2.0 from 2025/06/08

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 TEX interface

```
\documentclass{article}
\usepackage{color}
\usepackage{farbe}
\begin{document}
\definecolor{my-test-color}{rgb}{.2,0.85,1}

% Not a valid color: my-test-color
% \FarbeTextColor{my-test-color}{This text is set in my test color.}

\FarbeImport{my-test-color}

\FarbeTextColor{my-test-color}{This text is set in my test color.}
\end{document}
```

\FarbePdfLiteral

\FarbeColor
\FarbeColorEnd

```
This is not green.
\FarbeColor{green} This is green. It's still green.
\FarbeColorEnd It's not green.
```

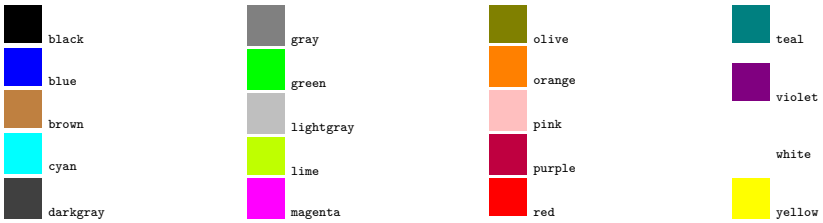
\FarbeTextColor

```
This is not green. \FarbeTextColor{green}{This is green.} This is not green.
```

\FarbeBox

4 Color names

4.1 base



4.2 svg

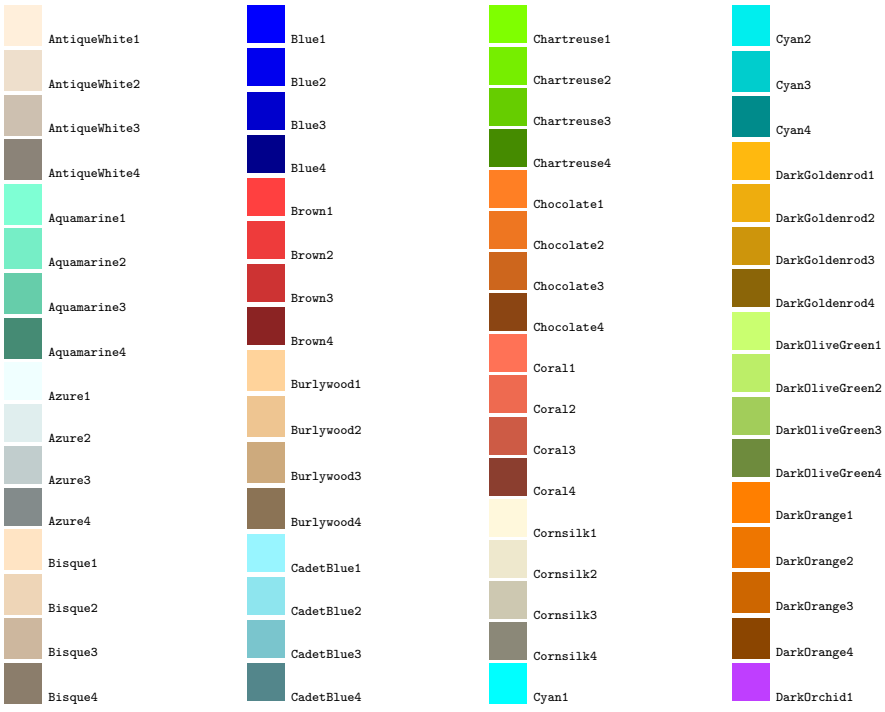
This svg color names are taken from the xcolor package.
Duplicate colors: `Aqua` = `Cyan`, `Fuchsia` = `Magenta`; `Navy` = `NavyBlue`; `Gray` = `Grey`, `DarkGray` = `DarkGrey`, `LightGray` = `LightGrey`, `SlateGray` = `SlateGrey`, `DarkSlateGray` = `DarkSlateGrey`, `LightSlateGray` = `LightSlateGrey`, `DimGray` = `DimGrey`.

HTML4 color keyword subset: `Aqua`, `Black`, `Blue`, `Fuchsia`, `Gray`, `Green`, `Lime`, `Maroon`, `Navy`, `Olive`, `Purple`, `Red`, `Silver`, `Teal`, `White`, `Yellow`.

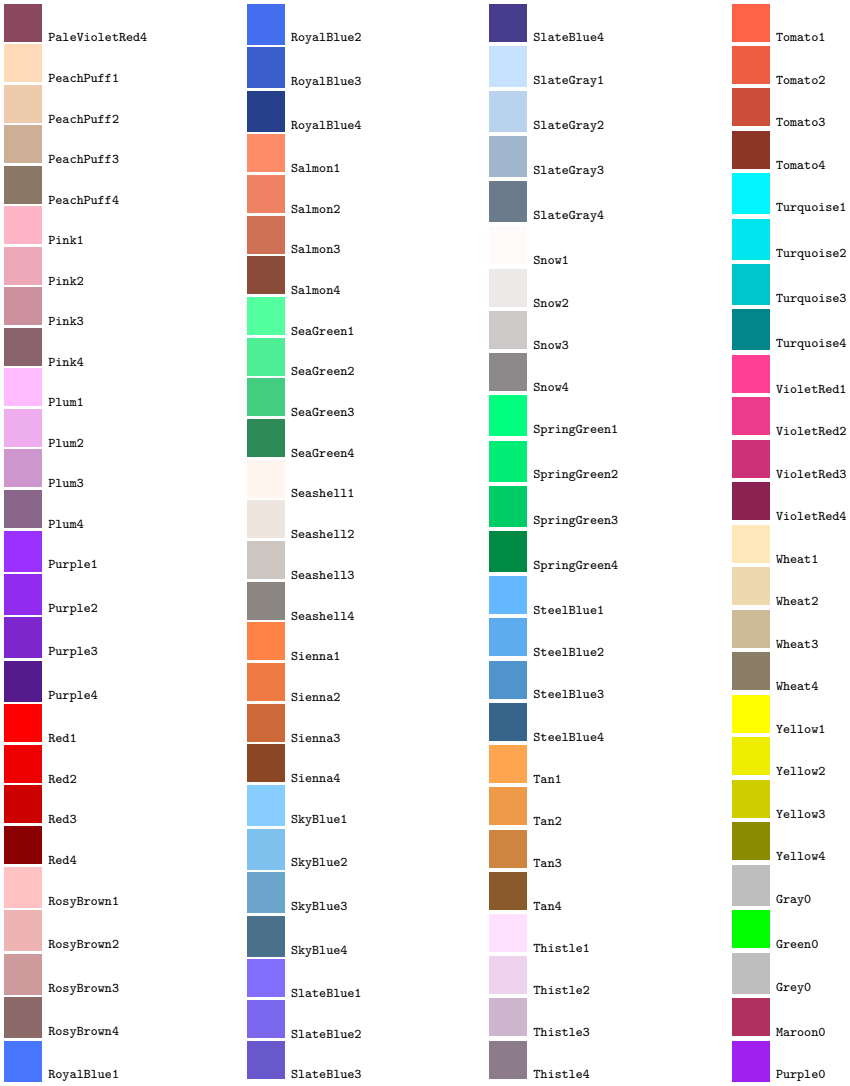
Colors taken from Unix/X11: `LightGoldenrod`, `LightSlateBlue`, `NavyBlue`, `VioletRed`.

4.3 x11

This x11 color names are taken from the xcolor package.
Duplicate colors: `Gray0` = `Grey0`,, `Green0` = `Green1`..




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

5 Implementation

5.1 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',
```

```
    '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',
```

```
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',
```

```
317         'HotPink4',
318         'IndianRed1',
319         'IndianRed2',
320         'IndianRed3',
321         'IndianRed4',
322         'Ivory1',
323         'Ivory2',
324         'Ivory3',
325         'Ivory4',
326         'Khaki1',
327         'Khaki2',
328         'Khaki3',
329         'Khaki4',
330         'LavenderBlush1',
331         'LavenderBlush2',
332         'LavenderBlush3',
333         'LavenderBlush4',
334         'LemonChiffon1',
335         'LemonChiffon2',
336         'LemonChiffon3',
337         'LemonChiffon4',
338         'LightBlue1',
339         'LightBlue2',
340         'LightBlue3',
341         'LightBlue4',
342         'LightCyan1',
343         'LightCyan2',
344         'LightCyan3',
345         'LightCyan4',
346         'LightGoldenrod1',
347         'LightGoldenrod2',
348         'LightGoldenrod3',
349         'LightGoldenrod4',
350         'LightPink1',
351         'LightPink2',
352         'LightPink3',
353         'LightPink4',
354         'LightSalmon1',
355         'LightSalmon2',
356         'LightSalmon3',
357         'LightSalmon4',
358         'LightSkyBlue1',
359         'LightSkyBlue2',
360         'LightSkyBlue3',
361         'LightSkyBlue4',
362         'LightSteelBlue1',
363         'LightSteelBlue2',
364         'LightSteelBlue3',
365         'LightSteelBlue4',
366         'LightYellow1',
367         'LightYellow2',
368         'LightYellow3',
369         'LightYellow4',
370         'Magenta1',
371         'Magenta2',
372         'Magenta3',
373         'Magenta4',
374         'Maroon1',
375         'Maroon2',
376         'Maroon3',
377         'Maroon4',
378         'MediumOrchid1',
379         'MediumOrchid2',
380         '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',
```

```
445         'RosyBrown4',
446         'RoyalBlue1',
447         'RoyalBlue2',
448         'RoyalBlue3',
449         'RoyalBlue4',
450         'Salmon1',
451         'Salmon2',
452         'Salmon3',
453         'Salmon4',
454         'SeaGreen1',
455         'SeaGreen2',
456         'SeaGreen3',
457         'SeaGreen4',
458         'Seashell1',
459         'Seashell2',
460         'Seashell3',
461         'Seashell4',
462         'Sienna1',
463         'Sienna2',
464         'Sienna3',
465         'Sienna4',
466         'SkyBlue1',
467         'SkyBlue2',
468         'SkyBlue3',
469         'SkyBlue4',
470         'SlateBlue1',
471         'SlateBlue2',
472         'SlateBlue3',
473         'SlateBlue4',
474         'SlateGray1',
475         'SlateGray2',
476         'SlateGray3',
477         'SlateGray4',
478         'Snow1',
479         'Snow2',
480         'Snow3',
481         'Snow4',
482         'SpringGreen1',
483         'SpringGreen2',
484         'SpringGreen3',
485         'SpringGreen4',
486         'SteelBlue1',
487         'SteelBlue2',
488         'SteelBlue3',
489         'SteelBlue4',
490         'Tan1',
491         'Tan2',
492         'Tan3',
493         'Tan4',
494         'Thistle1',
495         'Thistle2',
496         'Thistle3',
497         'Thistle4',
498         'Tomato1',
499         'Tomato2',
500         'Tomato3',
501         'Tomato4',
502         'Turquoise1',
503         'Turquoise2',
504         'Turquoise3',
505         'Turquoise4',
506         'VioletRed1',
507         'VioletRed2',
508         '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
    ---
    ---This svg color names are taken from the
    ↪  [xcolor](https://github.com/latex3/xcolor/blob/c5035d41c6070f4e8936196a994ad04336704872/xcolor.dtx#L7
    ↪  package.
    ---https://www.w3.org/TR/2003/REC-SVG11-20030114/types.html#ColorKeywords
    ---https://www.w3.org/TR/css-color-3/#svg-color
    ---
    AliceBlue = { .94, .972, 1 },
    AntiqueWhite = { .98, .92, .844 },
    Aqua = { 0, 1, 1 },
    Aquamarine = { .498, 1, .83 },
    Azure = { .94, 1, 1 },
    Beige = { .96, .96, .864 },
    Bisque = { 1, .894, .77 },
    Black = { 0, 0, 0 },
    BlanchedAlmond = { 1, .92, .804 },
    Blue = { 0, 0, 1 },
    BlueViolet = { .54, .17, .888 },
    Brown = { .648, .165, .165 },
    BurlyWood = { .87, .72, .53 },
    CadetBlue = { .372, .62, .628 },
    Chartreuse = { .498, 1, 0 },
    Chocolate = { .824, .41, .116 },
    Coral = { 1, .498, .312 },
    CornflowerBlue = { .392, .585, .93 },
    Cornsilk = { 1, .972, .864 },
    Crimson = { .864, .08, .235 },
    Cyan = { 0, 1, 1 },
    DarkBlue = { 0, 0, .545 },
    DarkCyan = { 0, .545, .545 },
    DarkGoldenrod = { .72, .525, .044 },
    DarkGray = { .664, .664, .664 },
    DarkGreen = { 0, .392, 0 },
    DarkGrey = { .664, .664, .664 },
    DarkKhaki = { .74, .716, .42 },
    DarkMagenta = { .545, 0, .545 },
    DarkOliveGreen = { .332, .42, .185 },
    DarkOrange = { 1, .55, 0 },
    DarkOrchid = { .6, .196, .8 },
    DarkRed = { .545, 0, 0 },
    DarkSalmon = { .912, .59, .48 },
    DarkSeaGreen = { .56, .736, .56 },
    DarkSlateBlue = { .284, .24, .545 },
    DarkSlateGray = { .185, .31, .31 },
    DarkSlateGrey = { .185, .31, .31 },
    DarkTurquoise = { 0, .808, .82 },
    DarkViolet = { .58, 0, .828 },
    DeepPink = { 1, .08, .576 },
    DeepSkyBlue = { 0, .75, 1 },
    DimGray = { .41, .41, .41 },
    DimGrey = { .41, .41, .41 },
    DodgerBlue = { .116, .565, 1 },
    FireBrick = { .698, .132, .132 },
    FloralWhite = { 1, .98, .94 },
    ForestGreen = { .132, .545, .132 },
    Fuchsia = { 1, 0, 1 },
    Gainsboro = { .864, .864, .864 },
    GhostWhite = { .972, .972, 1 },
```

```
    Gold = { 1, .844, 0 },
    Goldenrod = { .855, .648, .125 },
    Gray = { .5, .5, .5 },
    Green = { 0, .5, 0 },
    GreenYellow = { .68, 1, .185 },
    Grey = { .5, .5, .5 },
    Honeydew = { .94, 1, .94 },
    HotPink = { 1, .41, .705 },
    IndianRed = { .804, .36, .36 },
    Indigo = { .294, 0, .51 },
    Ivory = { 1, 1, .94 },
    Khaki = { .94, .9, .55 },
    Lavender = { .9, .9, .98 },
    LavenderBlush = { 1, .94, .96 },
    LawnGreen = { .488, .99, 0 },
    LemonChiffon = { 1, .98, .804 },
    LightBlue = { .68, .848, .9 },
    LightCoral = { .94, .5, .5 },
    LightCyan = { .88, 1, 1 },
    LightGoldenrod = { .933, .867, .51 }, -- Colors taken from Unix/X11
    LightGoldenrodYellow = { .98, .98, .824 },
    LightGray = { .828, .828, .828 },
    LightGreen = { .565, .932, .565 },
    LightGrey = { .828, .828, .828 },
    LightPink = { 1, .712, .756 },
    LightSalmon = { 1, .628, .48 },
    LightSeaGreen = { .125, .698, .668 },
    LightSkyBlue = { .53, .808, .98 },
    LightSlateBlue = { .518, .44, 1 }, -- Colors taken from Unix/X11
    LightSlateGray = { .468, .532, .6 },
    LightSlateGrey = { .468, .532, .6 },
    LightSteelBlue = { .69, .77, .87 },
    LightYellow = { 1, 1, .88 },
    Lime = { 0, 1, 0 },
    LimeGreen = { .196, .804, .196 },
    Linen = { .98, .94, .9 },
    Magenta = { 1, 0, 1 },
    Maroon = { .5, 0, 0 },
    MediumAquamarine = { .4, .804, .668 },
    MediumBlue = { 0, 0, .804 },
    MediumOrchid = { .73, .332, .828 },
    MediumPurple = { .576, .44, .86 },
    MediumSeaGreen = { .235, .7, .444 },
    MediumSlateBlue = { .484, .408, .932 },
    MediumSpringGreen = { 0, .98, .604 },
    MediumTurquoise = { .284, .82, .8 },
    MediumVioletRed = { .78, .084, .52 },
    MidnightBlue = { .098, .098, .44 },
    MintCream = { .96, 1, .98 },
    MistyRose = { 1, .894, .884 },
    Moccasin = { 1, .894, .71 },
    NavajoWhite = { 1, .87, .68 },
    Navy = { 0, 0, .5 },
    NavyBlue = { 0, 0, .5 }, -- Colors taken from Unix/X11
    OldLace = { .992, .96, .9 },
    Olive = { .5, .5, 0 },
    OliveDrab = { .42, .556, .136 },
    Orange = { 1, .648, 0 },
    OrangeRed = { 1, .27, 0 },
    Orchid = { .855, .44, .84 },
    PaleGoldenrod = { .932, .91, .668 },
    PaleGreen = { .596, .985, .596 },
    PaleTurquoise = { .688, .932, .932 },
    PaleVioletRed = { .86, .44, .576 },
```

```
    PapayaWhip = { 1, .936, .835 },
    PeachPuff = { 1, .855, .725 },
    Peru = { .804, .52, .248 },
    Pink = { 1, .752, .796 },
    Plum = { .868, .628, .868 },
    PowderBlue = { .69, .88, .9 },
    Purple = { .5, 0, .5 },
    Red = { 1, 0, 0 },
    RosyBrown = { .736, .56, .56 },
    RoyalBlue = { .255, .41, .884 },
    SaddleBrown = { .545, .27, .075 },
    Salmon = { .98, .5, .448 },
    SandyBrown = { .956, .644, .376 },
    SeaGreen = { .18, .545, .34 },
    Seashell = { 1, .96, .932 },
    Sienna = { .628, .32, .176 },
    Silver = { .752, .752, .752 },
    SkyBlue = { .53, .808, .92 },
    SlateBlue = { .415, .352, .804 },
    SlateGray = { .44, .5, .565 },
    SlateGrey = { .44, .5, .565 },
    Snow = { 1, .98, .98 },
    SpringGreen = { 0, 1, .498 },
    SteelBlue = { .275, .51, .705 },
    Tan = { .824, .705, .55 },
    Teal = { 0, .5, .5 },
    Thistle = { .848, .75, .848 },
    Tomato = { 1, .39, .28 },
    Turquoise = { .25, .88, .815 },
    Violet = { .932, .51, .932 },
    VioletRed = { .816, .125, .565 }, -- Colors taken from Unix/X11
    Wheat = { .96, .87, .7 },
    White = { 1, 1, 1 },
    WhiteSmoke = { .96, .96, .96 },
    Yellow = { 1, 1, 0 },
    YellowGreen = { .604, .804, .196 },

    ---x11
    ---
    ---This x11 color names are taken from the
    ↪  [xcolor](https://github.com/latex3/xcolor/blob/c5035d41c6070f4e8936196a994ad04336704872/xcolor.dtx#L72
    ↪  package.
    ---https://en.wikipedia.org/wiki/X11_color_names
    ---https://gitlab.freedesktop.org/xorg/app/rgb/raw/master/rgb.txt
    AntiqueWhite1 = { 1, .936, .86 },
    AntiqueWhite2 = { .932, .875, .8 },
    AntiqueWhite3 = { .804, .752, .69 },
    AntiqueWhite4 = { .545, .512, .47 },
    Aquamarine1 = { .498, 1, .83 },
    Aquamarine2 = { .464, .932, .776 },
    Aquamarine3 = { .4, .804, .668 },
    Aquamarine4 = { .27, .545, .455 },
    Azure1 = { .94, 1, 1 },
    Azure2 = { .88, .932, .932 },
    Azure3 = { .756, .804, .804 },
    Azure4 = { .512, .545, .545 },
    Bisque1 = { 1, .894, .77 },
    Bisque2 = { .932, .835, .716 },
    Bisque3 = { .804, .716, .62 },
    Bisque4 = { .545, .49, .42 },
    Blue1 = { 0, 0, 1 },
    Blue2 = { 0, 0, .932 },
    Blue3 = { 0, 0, .804 },
    Blue4 = { 0, 0, .545 },
```

```
    Brown1 = { 1, .25, .25 },
    Brown2 = { .932, .23, .23 },
    Brown3 = { .804, .2, .2 },
    Brown4 = { .545, .136, .136 },
    Burlywood1 = { 1, .828, .608 },
    Burlywood2 = { .932, .772, .57 },
    Burlywood3 = { .804, .668, .49 },
    Burlywood4 = { .545, .45, .332 },
    CadetBlue1 = { .596, .96, 1 },
    CadetBlue2 = { .556, .898, .932 },
    CadetBlue3 = { .48, .772, .804 },
    CadetBlue4 = { .325, .525, .545 },
    Chartreuse1 = { .498, 1, 0 },
    Chartreuse2 = { .464, .932, 0 },
    Chartreuse3 = { .4, .804, 0 },
    Chartreuse4 = { .27, .545, 0 },
    Chocolate1 = { 1, .498, .14 },
    Chocolate2 = { .932, .464, .13 },
    Chocolate3 = { .804, .4, .112 },
    Chocolate4 = { .545, .27, .075 },
    Coral1 = { 1, .448, .336 },
    Coral2 = { .932, .415, .312 },
    Coral3 = { .804, .356, .27 },
    Coral4 = { .545, .244, .185 },
    Cornsilk1 = { 1, .972, .864 },
    Cornsilk2 = { .932, .91, .804 },
    Cornsilk3 = { .804, .785, .694 },
    Cornsilk4 = { .545, .532, .47 },
    Cyan1 = { 0, 1, 1 },
    Cyan2 = { 0, .932, .932 },
    Cyan3 = { 0, .804, .804 },
    Cyan4 = { 0, .545, .545 },
    DarkGoldenrod1 = { 1, .725, .06 },
    DarkGoldenrod2 = { .932, .68, .055 },
    DarkGoldenrod3 = { .804, .585, .048 },
    DarkGoldenrod4 = { .545, .396, .03 },
    DarkOliveGreen1 = { .792, 1, .44 },
    DarkOliveGreen2 = { .736, .932, .408 },
    DarkOliveGreen3 = { .635, .804, .352 },
    DarkOliveGreen4 = { .43, .545, .24 },
    DarkOrange1 = { 1, .498, 0 },
    DarkOrange2 = { .932, .464, 0 },
    DarkOrange3 = { .804, .4, 0 },
    DarkOrange4 = { .545, .27, 0 },
    DarkOrchid1 = { .75, .244, 1 },
    DarkOrchid2 = { .698, .228, .932 },
    DarkOrchid3 = { .604, .196, .804 },
    DarkOrchid4 = { .408, .132, .545 },
    DarkSeaGreen1 = { .756, 1, .756 },
    DarkSeaGreen2 = { .705, .932, .705 },
    DarkSeaGreen3 = { .608, .804, .608 },
    DarkSeaGreen4 = { .41, .545, .41 },
    DarkSlateGray1 = { .592, 1, 1 },
    DarkSlateGray2 = { .552, .932, .932 },
    DarkSlateGray3 = { .475, .804, .804 },
    DarkSlateGray4 = { .32, .545, .545 },
    DeepPink1 = { 1, .08, .576 },
    DeepPink2 = { .932, .07, .536 },
    DeepPink3 = { .804, .064, .464 },
    DeepPink4 = { .545, .04, .312 },
    DeepSkyBlue1 = { 0, .75, 1 },
    DeepSkyBlue2 = { 0, .698, .932 },
    DeepSkyBlue3 = { 0, .604, .804 },
    DeepSkyBlue4 = { 0, .408, .545 },
```

```
    DodgerBlue1 = { .116, .565, 1 },
    DodgerBlue2 = { .11, .525, .932 },
    DodgerBlue3 = { .094, .455, .804 },
    DodgerBlue4 = { .064, .305, .545 },
    Firebrick1 = { 1, .19, .19 },
    Firebrick2 = { .932, .172, .172 },
    Firebrick3 = { .804, .15, .15 },
    Firebrick4 = { .545, .1, .1 },
    Gold1 = { 1, .844, 0 },
    Gold2 = { .932, .79, 0 },
    Gold3 = { .804, .68, 0 },
    Gold4 = { .545, .46, 0 },
    Goldenrod1 = { 1, .756, .145 },
    Goldenrod2 = { .932, .705, .132 },
    Goldenrod3 = { .804, .608, .112 },
    Goldenrod4 = { .545, .41, .08 },
    Green1 = { 0, 1, 0 },
    Green2 = { 0, .932, 0 },
    Green3 = { 0, .804, 0 },
    Green4 = { 0, .545, 0 },
    Honeydew1 = { .94, 1, .94 },
    Honeydew2 = { .88, .932, .88 },
    Honeydew3 = { .756, .804, .756 },
    Honeydew4 = { .512, .545, .512 },
    HotPink1 = { 1, .43, .705 },
    HotPink2 = { .932, .415, .655 },
    HotPink3 = { .804, .376, .565 },
    HotPink4 = { .545, .228, .385 },
    IndianRed1 = { 1, .415, .415 },
    IndianRed2 = { .932, .39, .39 },
    IndianRed3 = { .804, .332, .332 },
    IndianRed4 = { .545, .228, .228 },
    Ivory1 = { 1, 1, .94 },
    Ivory2 = { .932, .932, .88 },
    Ivory3 = { .804, .804, .756 },
    Ivory4 = { .545, .545, .512 },
    Khaki1 = { 1, .965, .56 },
    Khaki2 = { .932, .9, .52 },
    Khaki3 = { .804, .776, .45 },
    Khaki4 = { .545, .525, .305 },
    LavenderBlush1 = { 1, .94, .96 },
    LavenderBlush2 = { .932, .88, .898 },
    LavenderBlush3 = { .804, .756, .772 },
    LavenderBlush4 = { .545, .512, .525 },
    LemonChiffon1 = { 1, .98, .804 },
    LemonChiffon2 = { .932, .912, .75 },
    LemonChiffon3 = { .804, .79, .648 },
    LemonChiffon4 = { .545, .536, .44 },
    LightBlue1 = { .75, .936, 1 },
    LightBlue2 = { .698, .875, .932 },
    LightBlue3 = { .604, .752, .804 },
    LightBlue4 = { .408, .512, .545 },
    LightCyan1 = { .88, 1, 1 },
    LightCyan2 = { .82, .932, .932 },
    LightCyan3 = { .705, .804, .804 },
    LightCyan4 = { .48, .545, .545 },
    LightGoldenrod1 = { 1, .925, .545 },
    LightGoldenrod2 = { .932, .864, .51 },
    LightGoldenrod3 = { .804, .745, .44 },
    LightGoldenrod4 = { .545, .505, .298 },
    LightPink1 = { 1, .684, .725 },
    LightPink2 = { .932, .635, .68 },
    LightPink3 = { .804, .55, .585 },
    LightPink4 = { .545, .372, .396 },
```

```
    LightSalmon1 = { 1, .628, .48 },
    LightSalmon2 = { .932, .585, .448 },
    LightSalmon3 = { .804, .505, .385 },
    LightSalmon4 = { .545, .34, .26 },
    LightSkyBlue1 = { .69, .888, 1 },
    LightSkyBlue2 = { .644, .828, .932 },
    LightSkyBlue3 = { .552, .712, .804 },
    LightSkyBlue4 = { .376, .484, .545 },
    LightSteelBlue1 = { .792, .884, 1 },
    LightSteelBlue2 = { .736, .824, .932 },
    LightSteelBlue3 = { .635, .71, .804 },
    LightSteelBlue4 = { .43, .484, .545 },
    LightYellow1 = { 1, 1, .88 },
    LightYellow2 = { .932, .932, .82 },
    LightYellow3 = { .804, .804, .705 },
    LightYellow4 = { .545, .545, .48 },
    Magenta1 = { 1, 0, 1 },
    Magenta2 = { .932, 0, .932 },
    Magenta3 = { .804, 0, .804 },
    Magenta4 = { .545, 0, .545 },
    Maroon1 = { 1, .204, .7 },
    Maroon2 = { .932, .19, .655 },
    Maroon3 = { .804, .16, .565 },
    Maroon4 = { .545, .11, .385 },
    MediumOrchid1 = { .88, .4, 1 },
    MediumOrchid2 = { .82, .372, .932 },
    MediumOrchid3 = { .705, .32, .804 },
    MediumOrchid4 = { .48, .215, .545 },
    MediumPurple1 = { .67, .51, 1 },
    MediumPurple2 = { .624, .475, .932 },
    MediumPurple3 = { .536, .408, .804 },
    MediumPurple4 = { .365, .28, .545 },
    MistyRose1 = { 1, .894, .884 },
    MistyRose2 = { .932, .835, .824 },
    MistyRose3 = { .804, .716, .71 },
    MistyRose4 = { .545, .49, .484 },
    NavajoWhite1 = { 1, .87, .68 },
    NavajoWhite2 = { .932, .81, .63 },
    NavajoWhite3 = { .804, .7, .545 },
    NavajoWhite4 = { .545, .475, .37 },
    OliveDrab1 = { .752, 1, .244 },
    OliveDrab2 = { .7, .932, .228 },
    OliveDrab3 = { .604, .804, .196 },
    OliveDrab4 = { .41, .545, .132 },
    Orange1 = { 1, .648, 0 },
    Orange2 = { .932, .604, 0 },
    Orange3 = { .804, .52, 0 },
    Orange4 = { .545, .352, 0 },
    OrangeRed1 = { 1, .27, 0 },
    OrangeRed2 = { .932, .25, 0 },
    OrangeRed3 = { .804, .215, 0 },
    OrangeRed4 = { .545, .145, 0 },
    Orchid1 = { 1, .512, .98 },
    Orchid2 = { .932, .48, .912 },
    Orchid3 = { .804, .41, .79 },
    Orchid4 = { .545, .28, .536 },
    PaleGreen1 = { .604, 1, .604 },
    PaleGreen2 = { .565, .932, .565 },
    PaleGreen3 = { .488, .804, .488 },
    PaleGreen4 = { .33, .545, .33 },
    PaleTurquoise1 = { .732, 1, 1 },
    PaleTurquoise2 = { .684, .932, .932 },
    PaleTurquoise3 = { .59, .804, .804 },
    PaleTurquoise4 = { .4, .545, .545 },
```

```
    PaleVioletRed1 = { 1, .51, .67 },
    PaleVioletRed2 = { .932, .475, .624 },
    PaleVioletRed3 = { .804, .408, .536 },
    PaleVioletRed4 = { .545, .28, .365 },
    PeachPuff1 = { 1, .855, .725 },
    PeachPuff2 = { .932, .796, .68 },
    PeachPuff3 = { .804, .688, .585 },
    PeachPuff4 = { .545, .468, .396 },
    Pink1 = { 1, .71, .772 },
    Pink2 = { .932, .664, .72 },
    Pink3 = { .804, .57, .62 },
    Pink4 = { .545, .39, .424 },
    Plum1 = { 1, .732, 1 },
    Plum2 = { .932, .684, .932 },
    Plum3 = { .804, .59, .804 },
    Plum4 = { .545, .4, .545 },
    Purple1 = { .608, .19, 1 },
    Purple2 = { .57, .172, .932 },
    Purple3 = { .49, .15, .804 },
    Purple4 = { .332, .1, .545 },
    Red1 = { 1, 0, 0 },
    Red2 = { .932, 0, 0 },
    Red3 = { .804, 0, 0 },
    Red4 = { .545, 0, 0 },
    RosyBrown1 = { 1, .756, .756 },
    RosyBrown2 = { .932, .705, .705 },
    RosyBrown3 = { .804, .608, .608 },
    RosyBrown4 = { .545, .41, .41 },
    RoyalBlue1 = { .284, .464, 1 },
    RoyalBlue2 = { .264, .43, .932 },
    RoyalBlue3 = { .228, .372, .804 },
    RoyalBlue4 = { .152, .25, .545 },
    Salmon1 = { 1, .55, .41 },
    Salmon2 = { .932, .51, .385 },
    Salmon3 = { .804, .44, .33 },
    Salmon4 = { .545, .298, .224 },
    SeaGreen1 = { .33, 1, .624 },
    SeaGreen2 = { .305, .932, .58 },
    SeaGreen3 = { .264, .804, .5 },
    SeaGreen4 = { .18, .545, .34 },
    Seashell1 = { 1, .96, .932 },
    Seashell2 = { .932, .898, .87 },
    Seashell3 = { .804, .772, .75 },
    Seashell4 = { .545, .525, .51 },
    Sienna1 = { 1, .51, .28 },
    Sienna2 = { .932, .475, .26 },
    Sienna3 = { .804, .408, .224 },
    Sienna4 = { .545, .28, .15 },
    SkyBlue1 = { .53, .808, 1 },
    SkyBlue2 = { .494, .752, .932 },
    SkyBlue3 = { .424, .65, .804 },
    SkyBlue4 = { .29, .44, .545 },
    SlateBlue1 = { .512, .435, 1 },
    SlateBlue2 = { .48, .404, .932 },
    SlateBlue3 = { .41, .35, .804 },
    SlateBlue4 = { .28, .235, .545 },
    SlateGray1 = { .776, .888, 1 },
    SlateGray2 = { .725, .828, .932 },
    SlateGray3 = { .624, .712, .804 },
    SlateGray4 = { .424, .484, .545 },
    Snow1 = { 1, .98, .98 },
    Snow2 = { .932, .912, .912 },
    Snow3 = { .804, .79, .79 },
    Snow4 = { .545, .536, .536 },
```

```
    SpringGreen1 = { 0, 1, .498 },
    SpringGreen2 = { 0, .932, .464 },
    SpringGreen3 = { 0, .804, .4 },
    SpringGreen4 = { 0, .545, .27 },
    SteelBlue1 = { .39, .72, 1 },
    SteelBlue2 = { .36, .675, .932 },
    SteelBlue3 = { .31, .58, .804 },
    SteelBlue4 = { .21, .392, .545 },
    Tan1 = { 1, .648, .31 },
    Tan2 = { .932, .604, .288 },
    Tan3 = { .804, .52, .248 },
    Tan4 = { .545, .352, .17 },
    Thistle1 = { 1, .884, 1 },
    Thistle2 = { .932, .824, .932 },
    Thistle3 = { .804, .71, .804 },
    Thistle4 = { .545, .484, .545 },
    Tomato1 = { 1, .39, .28 },
    Tomato2 = { .932, .36, .26 },
    Tomato3 = { .804, .31, .224 },
    Tomato4 = { .545, .21, .15 },
    Turquoise1 = { 0, .96, 1 },
    Turquoise2 = { 0, .898, .932 },
    Turquoise3 = { 0, .772, .804 },
    Turquoise4 = { 0, .525, .545 },
    VioletRed1 = { 1, .244, .59 },
    VioletRed2 = { .932, .228, .55 },
    VioletRed3 = { .804, .196, .47 },
    VioletRed4 = { .545, .132, .32 },
    Wheat1 = { 1, .905, .73 },
    Wheat2 = { .932, .848, .684 },
    Wheat3 = { .804, .73, .59 },
    Wheat4 = { .545, .494, .4 },
    Yellow1 = { 1, 1, 0 },
    Yellow2 = { .932, .932, 0 },
    Yellow3 = { .804, .804, 0 },
    Yellow4 = { .545, .545, 0 },
    Gray0 = { .745, .745, .745 },
    Green0 = { 0, 1, 0 },
    Grey0 = { .745, .745, .745 },
    Maroon0 = { .69, .19, .376 },
    Purple0 = { .628, .125, .94 },
}

--- 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
```

```lua
    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

  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(
  --   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
```

```lua
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

  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)
```

```lua
---@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

---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
```

```lua

  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

```

```lua
  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

  ---
  ---@param h any # hue
  ---@param black any # black
  ---@param white any # white
  ---
  ---@return number r
  ---@return number g
  ---@return number b
  local function hwb_to_rgb(h, black, white)
    local r, g, b = hsv_to_rgb(h, 1, .5)
    local f = 1 - white - black
    return f * r + white, f * g + white, f * b + white
  end

  return {
    rgb_to_cmyk = rgb_to_cmyk,
    cmyk_to_rgb = cmyk_to_rgb,
    rgb_to_gray = rgb_to_gray,
    cmyk_to_gray = cmyk_to_gray,
    hsv_to_rgb = hsv_to_rgb,
    rgb_to_hsv = rgb_to_hsv,
    gray_to_rgb = gray_to_rgb,
    hsv_to_gray = hsv_to_gray,
    gray_to_hsv = gray_to_hsv,
    hwb_to_rgb = hwb_to_rgb,
  }

end)()

--- https://github.com/Firanel/lua-color/blob/master/init.lua

---The Class Color is the main class of the submodule. It represents a
---RGB color.
---
---@class Color
---@field r number # Red component.
---@field g number # Green component.
---@field b number # Blue component.
---@field a number # Alpha component.
---
---@function Color:__call
---
---@param value Color string|table|Color value (default: `nil`)
---
---@see Color:set
local Color = (function()

  ---Parse, convert and manipulate color values.
  ---
  -- @classmod Color

  -- Lua 5.1 compat
```

```lua
local bit_and = bitwise.bit_and
local bit_lshift = bitwise.bit_lshift
local bit_rshift = bitwise.bit_rshift

-- Utils

local function hcm_to_rgb(h, c, m)
  local r, g, b = 0, 0, 0

  h = h * 6
  local x = c * (1 - math.abs(h % 2 - 1))

  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

---
---@param str string A number encoded as a string with an optional percent sign.
---
---@return number result A number from 0 - 1
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)
```

```
1526      if value then
1527        if type(value) == 'string' then
1528          -- # gets expanded to ##
1529          value = string.gsub(value, '^##', '#')
1530        end
1531        this:set(value)
1532      end
1533    end, { __is_color = true, r = 0, g = 0, b = 0, a = 1 })
1534
1535    ---Clone color
1536    ---
1537    ---@return Color copy
1538    function Color:clone()
1539      return Color(self)
1540    end
1541
1542    ---Set color to value.
1543    -- <br>
1544    -- Called by constructor
1545    -- <br><br>
1546    -- Possible value types:
1547    -- <ul>
1548    --  <li>`Color`</li>
1549    --  <li>color name as specified in `Color.colorNames`</li>
1550    --  <li>css style functions as string:<ul>
1551    --   <li>`rgb(r, g, b)`</li>
1552    --   <li>`rgba(r, g, b, a)`</li>
1553    --   <li>`hsl(h, s, l)`</li>
1554    --   <li>`hsla(h, s, l, a)`</li>
1555    --   <li>`hsv(h, s, v)`</li>
1556    --   <li>`hsva(h, s, v, a)`</li>
1557    --   <li>`hwb(h, w, b)`</li>
1558    --   <li>`hwba(h, w, b, a)`</li>
1559    --   <li>`cmyk(c, m, y, k)`</li>
1560    --   </ul>
1561    --   Values are in the same ranges as in css ([0;255] for rgb, [0;1] for alpha,
      ↪  ...)<br>
1562    --   functions can be specified in a simplified syntax: `rgb(r, g, b) == rgb r g
      ↪  b`
1563    --  </li>
1564    --  <li>NCol string: `R10, 50%, 50%`</li>
1565    --  <li>hex string: `#rgb` | `#rgba` | `#rrggbb` | `#rrggbbaa` (`#` can be
      ↪  omitted)</li>
1566    --  <li>rgb values in [0;1]: `{r, g, b[, a]}` | `{r=r, g=g, b=b[, a=a]}`</li>
1567    --  <li>hsv values in [0;1]: `{h=h, s=s, v=v[, a=a]}`</li>
1568    --  <li>hsl values in [0;1]: `{h=h, s=s, l=l[, a=a]}`</li>
1569    --  <li>hwb values in [0;1]: `{h=h, w=w, b=b[, a=a]}`</li>
1570    --  <li>cmyk values in [0;1]: `{c=c, m=m, y=y, k=k}`</li>
1571    --  <li>single set mode, table with any combination of the following: <ul>
1572    --   <li>`red`</li>
1573    --   <li>`green`</li>
1574    --   <li>`blue`</li>
1575    --   <li>`alpha`</li>
1576    --   <li>`hue`</li>
1577    --   <li>`saturation`</li>
1578    --   <li>`value`</li>
1579    --   <li>`lightness`</li>
1580    --   <li>`whiteness`</li>
1581    --   <li>`blackness`</li>
1582    --   <li>`cyan`</li>
1583    --   <li>`magenta`</li>
1584    --   <li>`yellow`</li>
1585    --   <li>`key`</li>
1586    --   </ul>
```

```
1587    --   All values are in `[0;1]`.<br>
1588    --   They will be applied in the order: `rgba -> hsl -> hwb -> hsv -> cmyk`<br>
1589    --   If `lightness` is given, saturation is treated as hsl saturation,
1590    --   otherwise it will be treated as hsv saturation.
1591    --  </li>
1592    -- </ul>
1593    ---
1594    ---@see Color:__call
1595    ---
1596    ---@param value string|table|Color
1597    ---
1598    ---@return Color self
1599    ---
1600    ---@usage color:set "#f1f1f1"
1601    ---@usage color:set "rgba(241, 241, 241, 0.5)"
1602    ---@usage color:set "hsl 180 100% 20%"
1603    ---@usage color:set { r = 0.255, g = 0.729, b = 0.412 }
1604    ---@usage color:set { 0.255, 0.729, 0.412 } -- same as above
1605    ---@usage color:set { h = 0.389, s = 0.65, v = 0.73 }
1606    function Color:set(value)
1607      assert(value)
1608
1609      -- from Color
1610      if value.__is_color then
1611        self.r = value.r
1612        self.g = value.g
1613        self.b = value.b
1614        self.a = value.a
1615
1616      elseif type(value) == 'string' then
1617        self.a = 1
1618
1619        if value:sub(1, 1) ~= '#' then
1620          local c = colors[value]
1621          if c then
1622            self.r = c[1]
1623            self.g = c[2]
1624            self.b = c[3]
1625            return
1626          end
1627
1628          local func, values = value:match '(%w+)[ %(]+([x ,.%x%%]+)'
1629          if func ~= nil then
1630            if func == 'rgb' then
1631              local r, g, b =
1632                values:match '([x.%x]+)[ ,]+([x.%x]+)[ ,]+([x.%x]+)'
1633              assert(r and g and b)
1634              self.r = tonumber(r) / 0xff
1635              self.g = tonumber(g) / 0xff
1636              self.b = tonumber(b) / 0xff
1637              return self
1638            elseif func == 'rgba' then
1639              local r, g, b, a =
1640                values:match '([x.%x]+)[ ,]+([x.%x]+)[ ,]+([x.%x]+)[ ,]+([x.%x]+%%?)'
1641              assert(r and g and b and a)
1642              self.r = tonumber(r) / 0xff
1643              self.g = tonumber(g) / 0xff
1644              self.b = tonumber(b) / 0xff
1645              self.a = tonumPercent(a)
1646              return self
1647            elseif func == 'hsv' then
1648              local h, s, v =
1649                values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
1650              assert(h and s and v)
```

```
    return self:set{
      h = tonumber(h) / 360,
      s = tonumPercent(s),
      v = tonumPercent(v),
    }
  elseif func == 'hsva' then
    local h, s, v, a =
      values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
    assert(h and s and v and a)
    return self:set{
      h = tonumber(h) / 360,
      s = tonumPercent(s),
      v = tonumPercent(v),
      a = tonumPercent(a),
    }
  elseif func == 'hsl' then
    local h, s, l =
      values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
    assert(h and s and l)
    return self:set{
      h = tonumber(h) / 360,
      s = tonumPercent(s),
      l = tonumPercent(l),
    }
  elseif func == 'hsla' then
    local h, s, l, a =
      values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
    assert(h and s and l and a)
    return self:set{
      h = tonumber(h) / 360,
      s = tonumPercent(s),
      l = tonumPercent(l),
      a = tonumPercent(a),
    }
  elseif func == 'hwb' then
    local h, w, b =
      values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
    assert(h and w and b)
    return self:set{
      h = tonumber(h) / 360,
      w = tonumPercent(w),
      b = tonumPercent(b),
    }
  elseif func == 'hwba' then
    local h, w, b, a =
      values:match '([x.%x]+)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
    assert(h and w and b and a)
    return self:set{
      h = tonumber(h) / 360,
      w = tonumPercent(w),
      b = tonumPercent(b),
      a = tonumPercent(a),
    }
  elseif func == 'cmyk' then
    local c, m, y, k =
      values:match '([x.%x]+%%?)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
    assert(c and m and y and k)
    return self:set{
      c = tonumPercent(c),
      m = tonumPercent(m),
```

```
1711                y = tonumPercent(y),
1712                k = tonumPercent(k),
1713              }
1714            end
1715          else
1716            local col, dist, w, b, a =
1717              value:match '([RGBCMYrgbcmy])(%d*)[, ]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)[
                  ↪  ,]+([x.%x]+%%?)'
1718            if col == nil then
1719              col, dist, w, b, a =
1720                value:match '([RGBCMYrgbcmy])(%d*)[, ]+([x.%x]+%%?)[ ,]+([x.%x]+%%?)'
1721            end
1722            if col then
1723              col = col:lower()
1724
1725              local h
1726              if col == 'r' then
1727                h = 0
1728              elseif col == 'y' then
1729                h = 1 / 6
1730              elseif col == 'g' then
1731                h = 2 / 6
1732              elseif col == 'c' then
1733                h = 3 / 6
1734              elseif col == 'b' then
1735                h = 4 / 6
1736              elseif col == 'm' then
1737                h = 5 / 6
1738              end
1739
1740              if #dist > 0 then
1741                h = h + tonumber(dist) / 600
1742              end
1743
1744              return self:set{
1745                h = h,
1746                w = tonumPercent(w),
1747                b = tonumPercent(b),
1748                a = a and tonumPercent(a) or 1,
1749              }
1750            end
1751          end
1752        else
1753          value = value:sub(2)
1754        end
1755
1756        local pattern
1757        local div = 0xff
1758        if #value == 3 then
1759          pattern = '(%x)(%x)(%x)'
1760          div = 0xf
1761        elseif #value == 4 then
1762          pattern = '(%x)(%x)(%x)(%x)'
1763          div = 0xf
1764        elseif #value == 6 then
1765          pattern = '(%x%x)(%x%x)(%x%x)'
1766        elseif #value == 8 then
1767          pattern = '(%x%x)(%x%x)(%x%x)(%x%x)'
1768        else
1769          error('Not a valid color: ' .. tostring(value))
1770        end
1771        local r, g, b, a = value:match(pattern)
1772        assert(r ~= nil, 'Not a valid color: ' .. tostring(value))
1773        self.r = tonumber(r, 16) / div
```

```
1774          self.g = tonumber(g, 16) / div
1775          self.b = tonumber(b, 16) / div
1776          self.a = a ~= nil and tonumber(a, 16) / div or 1
1777
1778          -- table with rgb
1779        elseif value[1] ~= nil then
1780          self.r = value[1]
1781          self.g = value[2]
1782          self.b = value[3]
1783          self.a = value[4] or self.a or 1
1784        elseif value.r ~= nil then
1785          self.r = value.r
1786          self.g = value.g or self.g
1787          self.b = value.b or self.b
1788          self.a = value.a or self.a
1789
1790        elseif value.c ~= nil then
1791          self.r, self.g, self.b = convert.cmyk_to_rgb(value.c, value.m,
1792            value.y, value.k)
1793          self.a = 1
1794
1795          -- table with hs[vl]
1796        elseif value.h ~= nil then
1797          if value.w ~= nil then -- hwb
1798            value.v = 1 - value.b
1799            value.s = 1 - value.w / value.v
1800          end
1801
1802          local hue, saturation = value.h, value.s
1803          assert(hue ~= nil, saturation ~= nil)
1804
1805          local r, g, b = 0, 0, 0
1806
1807          if value.v ~= nil then
1808            local v = value.v
1809            local chroma = saturation * v
1810            r, g, b = hcm_to_rgb(hue, chroma, v - chroma)
1811
1812          elseif value.l ~= nil then
1813            local lightness = value.l
1814            local chroma = (1 - math.abs(2 * lightness - 1)) * saturation
1815            r, g, b = hcm_to_rgb(hue, chroma, lightness - chroma / 2)
1816          end
1817
1818          self.r = r
1819          self.g = g
1820          self.b = b
1821          self.a = value.a or self.a or 1
1822
1823        else -- Single set mode
1824          if value.red then
1825            self.r = value.red
1826          end
1827          if value.green then
1828            self.g = value.green
1829          end
1830          if value.blue then
1831            self.b = value.blue
1832          end
1833          if value.alpha then
1834            self.a = value.alpha
1835          end
1836
1837          if value.lightness then
```

```
1838          local h, s, l = self:hsl()
1839          self:set{
1840            h = value.hue or h,
1841            s = value.saturation or s,
1842            l = value.lightness or l,
1843          }
1844          value.hue = nil
1845          value.saturation = nil
1846        end
1847
1848        if value.whiteness or value.blackness then
1849          local h, w, b = self:hwb()
1850          self:set{
1851            h = value.hue or h,
1852            w = value.whiteness or w,
1853            b = value.backness or b,
1854          }
1855          value.hue = nil
1856        end
1857
1858        if value.hue or value.saturation or value.value then
1859          local h, s, v = self:hsv()
1860          self:set{
1861            h = value.hue or h,
1862            s = value.saturation or s,
1863            v = value.value or v,
1864          }
1865        end
1866
1867        if value.cyan or value.magenta or value.yellow or value.key then
1868          local c, m, y, k = self:cmyk()
1869          self:set{
1870            c = value.cyan or c,
1871            m = value.magenta or m,
1872            y = value.yellow or y,
1873            k = value.key or k,
1874          }
1875        end
1876      end
1877
1878      local r, g, b, a = utils.clamp(self.r, 0, 1),
1879        utils.clamp(self.g, 0, 1), utils.clamp(self.b, 0, 1),
1880        utils.clamp(self.a, 0, 1)
1881      assert(r and g and b and a, 'Color invalid')
1882      return self
1883    end
1884
1885    ---Get rgb values.
1886    ---
1887    ---@return number[0;1] red
1888    ---@return number[0;1] green
1889    ---@return number[0;1] blue
1890    function Color:rgb()
1891      return self.r, self.g, self.b
1892    end
1893
1894    ---Get rgba values.
1895    ---
1896    ---@return number[0;1] red
1897    ---@return number[0;1] green
1898    ---@return number[0;1] blue
1899    ---@return number[0;1] alpha
1900    function Color:rgba()
1901      return self.r, self.g, self.b, self.a
```

```lua
end

function Color:_hsvm()
  local r, g, b = self.r, self.g, self.b

  local max, max_i = utils.max(r, g, b)
  local min = math.min(r, g, b)
  local chroma = max - min

  local hue
  if chroma == 0 then
    hue = 0
  elseif max_i == 1 then
    hue = ((g - b) / chroma) / 6
  elseif max_i == 2 then
    hue = (2 + (b - r) / chroma) / 6
  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
```

```lua

---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
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
```

```
2029          error('No valid argument')
2030        end
2031
2032        local h, s, v = self:hsv()
2033        h = (h + r) % 1
2034        self:set{ h = h, s = s, v = v, a = self.a }
2035
2036        return self
2037      end
2038
2039      ---Invert the color.
2040      ---
2041      ---@return Color self
2042      function Color:invert()
2043        self.r = 1 - self.r
2044        self.g = 1 - self.g
2045        self.b = 1 - self.b
2046        return self
2047      end
2048
2049      ---Reduce saturation to 0.
2050      ---
2051      ---@return Color self
2052      function Color:grey()
2053        local h, _, v = self:hsv()
2054        self:set{ h = h, s = 0, v = v, a = self.a }
2055        return self
2056      end
2057
2058      ---Set to black or white depending on lightness.
2059      ---
2060      ---@param ?number[0;1] lightness Cutoff point (Default: 0.5)
2061      ---
2062      ---@return Color self
2063      function Color:blackOrWhite(lightness)
2064        local _, _, l = self:hsl()
2065        local v = l > lightness and 1 or 0
2066        self.r = v
2067        self.g = v
2068        self.b = v
2069        return self
2070      end
2071
2072      ---Mix two colors together.
2073      ---
2074      ---@param Color other
2075      ---@param ?number strength 0 results in self, 1 results in other (Default: 0.5)
2076      ---
2077      ---@return Color self
2078      function Color:mix(other, strength)
2079        if strength == nil then
2080          strength = 0.5
2081        end
2082        self.r = self.r * (1 - strength) + other.r * strength
2083        self.g = self.g * (1 - strength) + other.g * strength
2084        self.b = self.b * (1 - strength) + other.b * strength
2085        self.a = self.a * (1 - strength) + other.a * strength
2086        return self
2087      end
2088
2089      ---Generate complementary color.
2090      ---
2091      ---@return Color
2092      function Color:complement()
```

```lua
  return Color(self):rotate(0.5)
end

---Generate analogous color scheme.
---
---@return Color
---@return Color self
---@return Color
function Color:analogous()
  local h, s, v = self:hsv()
  return Color { h = (h - 1 / 12) % 1, s = s, v = v, a = self.a },
    self, Color { h = (h + 1 / 12) % 1, s = s, v = v, a = self.a }
end

---Generate triadic color scheme.
---
---@return Color self
---@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 }

```

```lua
    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`,
  --  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
```

```
2221        local h, s, l = self:hsl()
2222        return string.format('hsl(%d, %d%%, %d%%)', utils.round(h * 360),
2223          utils.round(s * 100), utils.round(l * 100))
2224      elseif format == 'hsla' then
2225        local h, s, l, a = self:hsla()
2226        return string.format('hsla(%d, %d%%, %d%%, %s)',
2227          utils.round(h * 360), utils.round(s * 100),
2228          utils.round(l * 100), a)
2229      elseif format == 'hwb' then
2230        local h, w, b = self:hwb()
2231        return string.format('hwb(%d, %d%%, %d%%)', utils.round(h * 360),
2232          utils.round(w * 100), utils.round(b * 100))
2233      elseif format == 'hwba' then
2234        local h, w, b, a = self:hwba()
2235        return string.format('hwba(%d, %d%%, %d%%, %s)',
2236          utils.round(h * 360), utils.round(w * 100),
2237          utils.round(b * 100), a)
2238      elseif format == 'ncol' then
2239        local h, w, b = self:hwb()
2240        local h_maj, h_min = math.modf(h * 6)
2241        h_maj = h_maj % 6
2242
2243        local col
2244        if h_maj == 0 then
2245          col = 'R'
2246        elseif h_maj == 1 then
2247          col = 'Y'
2248        elseif h_maj == 2 then
2249          col = 'G'
2250        elseif h_maj == 3 then
2251          col = 'C'
2252        elseif h_maj == 4 then
2253          col = 'B'
2254        else
2255          col = 'M'
2256        end
2257
2258        return string.format('%s%d, %d%%, %d%%', col,
2259          utils.round(h_min * 100), utils.round(w * 100),
2260          utils.round(b * 100))
2261      elseif format == 'cmyk' then
2262        local c, m, y, k = self:cmyk()
2263        return string.format('cymk(%d%%, %d%%, %d%%, %d%%)',
2264          utils.round(c * 100), utils.round(m * 100),
2265          utils.round(y * 100), utils.round(k * 100))
2266      end
2267
2268      return tostring(self)
2269    end
2270
2271    ---Get color in rgb hex notation.
2272    -- <br>
2273    -- only adds alpha value if `color.a < 1`
2274    ---
2275    ---@return string `#rrggbb` | `#rrggbbaa`
2276    ---
2277    ---@see Color:tostring
2278    function Color:__tostring()
2279      if self.a < 1 then
2280        return string.format('#%02x%02x%02x%02x',
2281          utils.round(self.r * 0xff), utils.round(self.g * 0xff),
2282          utils.round(self.b * 0xff), utils.round(self.a * 0xff))
2283      else
2284        return string.format('#%02x%02x%02x', utils.round(self.r * 0xff),
```

```
2285            utils.round(self.g * 0xff), utils.round(self.b * 0xff))
2286        end
2287      end
2288
2289      ---Check if colors are equal.
2290      ---
2291      ---@param Color other
2292      ---
2293      ---@return boolean all values are equal
2294      function Color:__eq(other)
2295        return
2296          self.r == other.r and self.g == other.g and self.b == other.b and
2297            self.a == other.a
2298      end
2299
2300      ---Checks whether color is darker.
2301      ---
2302      ---@param Color other
2303      ---
2304      ---@return boolean self is darker than other
2305      function Color:__lt(other)
2306        local _, _, la = self:hsl()
2307        local _, _, lb = other:hsl()
2308        return la < lb
2309      end
2310
2311      ---Checks whether color is as dark or darker.
2312      ---
2313      ---@param Color other
2314      ---
2315      ---@return boolean self is as dark or darker than other
2316      function Color:__le(other)
2317        local _, _, la = self:hsl()
2318        local _, _, lb = other:hsl()
2319        return la <= lb
2320      end
2321
2322      ---Iterate through color.
2323      ---
2324      -- Iterates through r, g, b, and a.
2325      function Color:__pairs()
2326        local function iter(tbl, k)
2327          if k == nil then
2328            return 'r', self.r
2329          elseif k == 'r' then
2330            return 'g', self.g
2331          elseif k == 'g' then
2332            return 'b', self.b
2333          elseif k == 'b' then
2334            return 'a', self.a
2335          end
2336        end
2337
2338        return iter, self, nil
2339      end
2340
2341      ---Get inverted clone of color.
2342      ---
2343      ---@return Color
2344      function Color:__unm()
2345        return Color(self):invert()
2346      end
2347
2348      ---Mix two colors evenly.
```

```
2349    ---
2350    ---@param Color a first color
2351    ---@param Color b second color
2352    ---
2353    ---@return Color new color
2354    ---
2355    ---@see Color:mix
2356    function Color.__add(a, b)
2357      assert(Color.isColor(a) and Color.isColor(b),
2358        'Can only add two colors.')
2359      return Color(a):mix(b)
2360    end
2361
2362    ---Complement of even mix.
2363    ---
2364    ---@param Color a first color
2365    ---@param Color b second color
2366    ---
2367    ---@return Color new color
2368    ---
2369    ---@see Color:mix
2370    ---@see Color.__add
2371    function Color.__sub(a, b)
2372      assert(Color.isColor(a) and Color.isColor(b),
2373        'Can only add two colors.')
2374      return Color(a):mix(b):rotate(0.5)
2375    end
2376
2377    ---Apply rgb mask to color.
2378    ---
2379    ---@param Color|number a color or mask
2380    ---@param Color|number b color or mask (if a and b are colors b is used as mask)
2381    ---
2382    ---@return Color new color
2383    ---
2384    ---@usage local new_col = color & 0xff00ff -- get new color without the green
       ↪  channel
2385    function Color.__band(a, b)
2386      local color, mask
2387      if Color.isColor(a) and type(b) == 'number' then
2388        color = a
2389        mask = b
2390      elseif Color.isColor(b) and type(a) == 'number' then
2391        color = b
2392        mask = a
2393      elseif Color.isColor(a) and Color.isColor(b) then
2394        color = a
2395        mask = bit_lshift(utils.round(b.r * 0xff), 16) +
2396                 bit_lshift(utils.round(b.g * 0xff), 8) +
2397                 utils.round(b.b * 0xff)
2398      else
2399        error(
2400          'Required arguments: Color|number,Color|number Received: ' ..
2401            type(a) .. ',' .. type(b))
2402      end
2403
2404      return Color {
2405        bit_and(utils.round(color.r * 0xff), bit_rshift(mask, 16)) / 0xff,
2406        bit_and(utils.round(color.g * 0xff), bit_rshift(mask, 8)) / 0xff,
2407        bit_and(utils.round(color.b * 0xff), mask) / 0xff,
2408        color.a,
2409      }
2410    end
2411
```

```lua
---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.
---
---@param a Color|number # color or mask
---@param b Color|number # color or mask (if a and b are colors b is used as mask)
---
---@return Color new color
---
---@usage local new_col = Color.band(color, 0xff00ff) -- get new color without the green channel
function Color.band(a, b)
  return Color.__band(a, b)
end

---Check whether `color` is a Color.
---
---@param color Color
---
---@return boolean # is a color
---
---@usage if Color.isColor(color) then print "It's a color!" end
function Color.isColor(color)
  return color ~= nil and color.__is_color == true
end

---Format a PDF colorstack string. This string can be assigned to the
---`node.data` field of a PDF colorstack node.
---
---@return string # A string like this example `1 0 0 rg 1 0 0 RG`
function Color:format_pdf_colorstack_string()
  return table.concat({
    self.r,
    self.g,
    self.b,
    'rg',
    self.r,
    self.g,
    self.b,
    'RG',
  }, ' ')
end

---Create a PDF colorstack node.
---
---@param command "set"|"push"|"pop"|"current"
---
---@return PdfColorstackWhatsitNode
function Color:create_pdf_colorstack_node(command)
  local whatsit = node.new('whatsit', 'pdf_colorstack') --[[@as PdfColorstackWhatsitNode]]
  if command == 'set' then
    whatsit.command = 0
  elseif command == 'push' then
    whatsit.command = 1
  elseif command == 'pop' then
    whatsit.command = 2
  elseif command == 'current' then
    whatsit.command = 3
  end
  if command ~= 'pop' then
    whatsit.data = self:format_pdf_colorstack_string()
  end
  return whatsit
```

```
  end

  ---Write a PDF colorstock node using `node.write()`.
  ---
  ---@param command "set"|"push"|"pop"|"current"
  ---
  function Color:write_pdf_colorstack_node(command)
    node.write(self:create_pdf_colorstack_node(command))
  end

  function Color:write_box()
    self:write_pdf_colorstack_node('push')
    local rule = node.new('rule') --[[@as RuleNode]]
    rule.width = tex.sp('0.5cm')
    rule.height = tex.sp('0.5cm')
    node.write(rule)
    self:write_pdf_colorstack_node('pop')
  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{rgb(%d, %d, %d} ', r, g, b))
    tex.print('\\texttt{\\tiny\\enspace ' .. 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
```

```
2535
2536    local n = tonumber
2537
2538    local r, g, b = operator:match(build_pattern(3, 'rg'))
2539    if r ~= nil then
2540      log.debug('operator to RGB: %s %s %s', r, g, b)
2541      return Color({ r = n(r), g = n(g), b = n(b) })
2542    end
2543
2544    local c, m, y, k = operator:match(build_pattern(3, 'k'))
2545    if c ~= nil then
2546      log.debug('operator to CMYK: %s %s %s %s', c, m, y, k)
2547      return Color({ c = n(c), m = n(m), y = n(y), k = n(k) })
2548    end
2549
2550    local gray = operator:match(build_pattern(1, 'g'))
2551    if gray ~= nil then
2552      log.debug('operator to GRAY: %s', gray)
2553      return Color({ r = n(gray), g = n(gray), b = n(gray) })
2554    end
2555  end
2556
2557  ---
2558  ---@param name string # The name of the color.
2559  ---@param operator string # The PDF color operator, e. g. `0.2 0.5 1 rg 0.2 0.5 1
     ↪  RG`
2560  local function import_color(name, operator)
2561    if colors[name] ~= nil then
2562      return
2563    end
2564    local color = convert_pdf_color_operator(operator)
2565    log.info('Import new color: name %s, operator: %s, converted: %s',
2566      name, operator, color)
2567    if color ~= nil then
2568      colors[name] = { color.r, color.g, color.b }
2569    end
2570  end
2571
2572  return {
2573    convert = convert,
2574    Color = Color --[[@as Color]] ,
2575    print_color_table = print_color_table,
2576    import_color = import_color,
2577  }
```

5.2 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\FarbeColorEnd
{%
  \directlua{farbe.Color('000'):write_pdf_colorstack_node('pop')}%
}

\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()}%
}
```

5.3 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}[2025/06/08 v0.2.0 Color management (conversion, names) for
↪  LuaTeX implemented in Lua.]

\input farbe.tex
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.