

The **farbe** package

Josef Friedrich

josef@friedrich.rocks

github.com/Josef-Friedrich/farbe

0.1.0 from 2025/05/31

Contents

1	Introduction	3
2	Lua interface / API	3
2.1	Class “Color”	3
2.1.1	Fields	3
2.1.1.1	Color.r	3
2.1.1.2	Color.g	3
2.1.1.3	Color.b	3
2.1.1.4	Color.a	3
2.1.2	Methods	3
2.1.2.1	Color:clone ()	3
2.1.2.2	Color:set (value)	3
2.1.2.3	Color:rgb ()	3
2.1.2.4	Color:rgba ()	3
2.1.2.5	Color:hsv ()	3
2.1.2.6	Color:hsva ()	3
2.1.2.7	Color:hsl ()	3
2.1.2.8	Color:hsla ()	3
2.1.2.9	Color:hwb ()	3
2.1.2.10	Color:hwba ()	3
2.1.2.11	Color:cmyk ()	3
2.1.2.12	Color:rotate (value)	3
2.1.2.13	Color:invert ()	4
2.1.2.14	Color:grey ()	4
2.1.2.15	Color:blackOrWhite (lightness)	4
2.1.2.16	Color:mix (other, strength)	4
2.1.2.17	Color:complement ()	4
2.1.2.18	Color:analogous ()	4
2.1.2.19	Color:triad ()	4
2.1.2.20	Color:tetrad ()	4
2.1.2.21	Color:compound ()	4
2.1.2.22	Color:evenlySpaced (n, r)	4
2.1.2.23	Color:tostring (format)	4
2.1.2.24	Color:band (a, b)	4
2.1.2.25	Color:isColor (color)	4
3	Color names	5
3.1	base	5
3.2	svg	5
3.3	x11	9
4	Implementation	19
4.1	farbe.lua	19
4.2	farbe.tex	60
4.3	farbe.sty	61

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:cmymk ()** 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

```
1  -- farbe.lua
2  -- Copyright 2025 Josef Friedrich
3  --
4  -- This work may be distributed and/or modified under the
5  -- conditions of the LaTeX Project Public License, either version 1.3c
6  -- of this license or (at your option) any later version.
7  -- The latest version of this license is in
8  -- http://www.latex-project.org/lppl.txt
9  -- and version 1.3c or later is part of all distributions of LaTeX
10 -- version 2008/05/04 or later.
11 --
12 -- This work has the LPPL maintenance status `maintained'.
13 --
14 -- The Current Maintainer of this work is Josef Friedrich.
15 --
16 -- This work consists of the files farbe.lua, farbe.tex,
17 -- and farbe.sty.
18 -- https://github.com/latex3/xcolor/blob/main/xcolor.dtx
19 ---@alias r number red (0.0 - 1.0)
20 ---@alias g number green (0.0 - 1.0)
21 ---@alias b number blue (0.0 - 1.0)
22 ---@alias a number alpha (0.0 - 1.0)
23 ---@alias c number cyan (0.0 - 1.0)
24 ---@alias m number magenta (0.0 - 1.0)
25 ---@alias y number yellow (0.0 - 1.0)
26 ---@alias k number key(black) (0.0 - 1.0)
27 local schemes = {
28
29   base = {
30     'black',
31     'blue',
32     'brown',
33     'cyan',
34     'darkgray',
35     'gray',
36     'green',
37     'lightgray',
38     'lime',
39     'magenta',
40     'olive',
41     'orange',
42     'pink',
43     'purple',
44     'red',
45     'teal',
46     'violet',
47     'white',
48     'yellow',
49   },
50
51   svg = {
52     'AliceBlue',
53     'AntiqueWhite',
54     'Aqua',
55     'Aquamarine',
56     'Azure',
57     'Beige',
58     'Bisque',
59     'Black',
60     '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',

```

189     'SpringGreen',
190     'SteelBlue',
191     'Tan',
192     'Teal',
193     'Thistle',
194     'Tomato',
195     'Turquoise',
196     'Violet',
197     'VioletRed',
198     'Wheat',
199     'White',
200     'WhiteSmoke',
201     'Yellow',
202     'YellowGreen',
203 },
204
205 x11 = {
206     'AntiqueWhite1',
207     'AntiqueWhite2',
208     'AntiqueWhite3',
209     'AntiqueWhite4',
210     'Aquamarine1',
211     'Aquamarine2',
212     'Aquamarine3',
213     'Aquamarine4',
214     'Azure1',
215     'Azure2',
216     'Azure3',
217     'Azure4',
218     'Bisque1',
219     'Bisque2',
220     'Bisque3',
221     'Bisque4',
222     'Blue1',
223     'Blue2',
224     'Blue3',
225     'Blue4',
226     'Brown1',
227     'Brown2',
228     'Brown3',
229     'Brown4',
230     'Burlywood1',
231     'Burlywood2',
232     'Burlywood3',
233     'Burlywood4',
234     'CadetBlue1',
235     'CadetBlue2',
236     'CadetBlue3',
237     'CadetBlue4',
238     'Chartreuse1',
239     'Chartreuse2',
240     'Chartreuse3',
241     'Chartreuse4',
242     'Chocolate1',
243     'Chocolate2',
244     'Chocolate3',
245     'Chocolate4',
246     'Coral1',
247     'Coral2',
248     'Coral3',
249     'Coral4',
250     'Cornsilk1',
251     'Cornsilk2',
252     'Cornsilk3',

```

```
253 'Cornsilk4',
254 'Cyan1',
255 'Cyan2',
256 'Cyan3',
257 'Cyan4',
258 'DarkGoldenrod1',
259 'DarkGoldenrod2',
260 'DarkGoldenrod3',
261 'DarkGoldenrod4',
262 'DarkOliveGreen1',
263 'DarkOliveGreen2',
264 'DarkOliveGreen3',
265 'DarkOliveGreen4',
266 'DarkOrange1',
267 'DarkOrange2',
268 'DarkOrange3',
269 'DarkOrange4',
270 'DarkOrchid1',
271 'DarkOrchid2',
272 'DarkOrchid3',
273 'DarkOrchid4',
274 'DarkSeaGreen1',
275 'DarkSeaGreen2',
276 'DarkSeaGreen3',
277 'DarkSeaGreen4',
278 'DarkSlateGray1',
279 'DarkSlateGray2',
280 'DarkSlateGray3',
281 'DarkSlateGray4',
282 'DeepPink1',
283 'DeepPink2',
284 'DeepPink3',
285 'DeepPink4',
286 'DeepSkyBlue1',
287 'DeepSkyBlue2',
288 'DeepSkyBlue3',
289 'DeepSkyBlue4',
290 'DodgerBlue1',
291 'DodgerBlue2',
292 'DodgerBlue3',
293 'DodgerBlue4',
294 'Firebrick1',
295 'Firebrick2',
296 'Firebrick3',
297 'Firebrick4',
298 'Gold1',
299 'Gold2',
300 'Gold3',
301 'Gold4',
302 'Goldenrod1',
303 'Goldenrod2',
304 'Goldenrod3',
305 'Goldenrod4',
306 'Green1',
307 'Green2',
308 'Green3',
309 'Green4',
310 'Honeydew1',
311 'Honeydew2',
312 'Honeydew3',
313 'Honeydew4',
314 'HotPink1',
315 'HotPink2',
316 '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',
```


381 'MediumOrchid4',
382 'MediumPurple1',
383 'MediumPurple2',
384 'MediumPurple3',
385 'MediumPurple4',
386 'MistyRose1',
387 'MistyRose2',
388 'MistyRose3',
389 'MistyRose4',
390 'NavajoWhite1',
391 'NavajoWhite2',
392 'NavajoWhite3',
393 'NavajoWhite4',
394 'OliveDrab1',
395 'OliveDrab2',
396 'OliveDrab3',
397 'OliveDrab4',
398 'Orange1',
399 'Orange2',
400 'Orange3',
401 'Orange4',
402 'OrangeRed1',
403 'OrangeRed2',
404 'OrangeRed3',
405 'OrangeRed4',
406 'Orchid1',
407 'Orchid2',
408 'Orchid3',
409 'Orchid4',
410 'PaleGreen1',
411 'PaleGreen2',
412 'PaleGreen3',
413 'PaleGreen4',
414 'PaleTurquoise1',
415 'PaleTurquoise2',
416 'PaleTurquoise3',
417 'PaleTurquoise4',
418 'PaleVioletRed1',
419 'PaleVioletRed2',
420 'PaleVioletRed3',
421 'PaleVioletRed4',
422 'PeachPuff1',
423 'PeachPuff2',
424 'PeachPuff3',
425 'PeachPuff4',
426 'Pink1',
427 'Pink2',
428 'Pink3',
429 'Pink4',
430 'Plum1',
431 'Plum2',
432 'Plum3',
433 'Plum4',
434 'Purple1',
435 'Purple2',
436 'Purple3',
437 'Purple4',
438 'Red1',
439 'Red2',
440 'Red3',
441 'Red4',
442 'RosyBrown1',
443 'RosyBrown2',
444 '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',
```

```

509     'VioletRed4',
510     'Wheat1',
511     'Wheat2',
512     'Wheat3',
513     'Wheat4',
514     'Yellow1',
515     'Yellow2',
516     'Yellow3',
517     'Yellow4',
518     'Gray0',
519     'Green0',
520     'Grey0',
521     'Maroon0',
522     'Purple0',
523 },
524 }
525
526 local log = (function()
527     local opts = { verbosity = 0 }
528
529     local function print_message(message, ...)
530         print(string.format(message, ...))
531     end
532
533     local function info(message, ...)
534         if opts.verbosity > 0 then
535             print_message(message, ...)
536         end
537     end
538
539     local function debug(message, ...)
540         if opts.verbosity > 1 then
541             print_message(message, ...)
542         end
543     end
544
545     local function verbose(message, ...)
546         if opts.verbosity > 2 then
547             print_message(message, ...)
548         end
549     end
550
551     return { opts = opts, info = info, debug = debug, verbose = verbose }
552 end)()
553
554 log.opts.verbosity = 3
555
556 local colors = {
557     -- base
558     black = { 0, 0, 0 },
559     blue = { 0, 0, 1 },
560     brown = { 0.75, 0.5, 0.25 },
561     cyan = { 0, 1, 1 },
562     darkgray = { 0.25, 0.25, 0.25 },
563     gray = { 0.5, 0.5, 0.5 },
564     green = { 0, 1, 0 },
565     lightgray = { 0.75, 0.75, 0.75 },
566     lime = { 0.75, 1, 0 },
567     magenta = { 1, 0, 1 },
568     olive = { 0.5, 0.5, 0 },
569     orange = { 1, 0.5, 0 },
570     pink = { 1, 0.75, 0.75 },
571     purple = { 0.75, 0, 0.25 },
572     red = { 1, 0, 0 },

```

```

573 teal = { 0, 0.5, 0.5 },
574 violet = { 0.5, 0, 0.5 },
575 white = { 1, 1, 1 },
576 yellow = { 1, 1, 0 },
577
578 -- sug
579 AliceBlue = { 0.94, 0.972, 0.972 },
580 AntiqueWhite = { 0.98, 0.92, 0.92 },
581 Aqua = { 0, 1, 1 },
582 Aquamarine = { 0.498, 1, 1 },
583 Azure = { 0.94, 1, 1 },
584 Beige = { 0.96, 0.96, 0.96 },
585 Bisque = { 1, 0.894, 0.894 },
586 Black = { 0, 0, 0 },
587 BlanchedAlmond = { 1, 0.92, 0.92 },
588 Blue = { 0, 0, 0 },
589 BlueViolet = { 0.54, 0.17, 0.17 },
590 Brown = { 0.648, 0.165, 0.165 },
591 BurlyWood = { 0.87, 0.72, 0.72 },
592 CadetBlue = { 0.372, 0.62, 0.62 },
593 Chartreuse = { 0.498, 1, 1 },
594 Chocolate = { 0.824, 0.41, 0.41 },
595 Coral = { 1, 0.498, 0.498 },
596 CornflowerBlue = { 0.392, 0.585, 0.585 },
597 Cornsilk = { 1, 0.972, 0.972 },
598 Crimson = { 0.864, 0.08, 0.08 },
599 Cyan = { 0, 1, 1 },
600 DarkBlue = { 0, 0, 0 },
601 DarkCyan = { 0, 0.545, 0.545 },
602 DarkGoldenrod = { 0.72, 0.525, 0.525 },
603 DarkGray = { 0.664, 0.664, 0.664 },
604 DarkGreen = { 0, 0.392, 0.392 },
605 DarkGrey = { 0.664, 0.664, 0.664 },
606 DarkKhaki = { 0.74, 0.716, 0.716 },
607 DarkMagenta = { 0.545, 0, 0 },
608 DarkOliveGreen = { 0.332, 0.42, 0.42 },
609 DarkOrange = { 1, 0.55, 0.55 },
610 DarkOrchid = { 0.6, 0.196, 0.196 },
611 DarkRed = { 0.545, 0, 0 },
612 DarkSalmon = { 0.912, 0.59, 0.59 },
613 DarkSeaGreen = { 0.56, 0.736, 0.736 },
614 DarkSlateBlue = { 0.284, 0.24, 0.24 },
615 DarkSlateGray = { 0.185, 0.31, 0.31 },
616 DarkSlateGrey = { 0.185, 0.31, 0.31 },
617 DarkTurquoise = { 0, 0.808, 0.808 },
618 DarkViolet = { 0.58, 0, 0 },
619 DeepPink = { 1, 0.08, 0.08 },
620 DeepSkyBlue = { 0, 0.75, 0.75 },
621 DimGray = { 0.41, 0.41, 0.41 },
622 DimGrey = { 0.41, 0.41, 0.41 },
623 DodgerBlue = { 0.116, 0.565, 0.565 },
624 FireBrick = { 0.698, 0.132, 0.132 },
625 FloralWhite = { 1, 0.98, 0.98 },
626 ForestGreen = { 0.132, 0.545, 0.545 },
627 Fuchsia = { 1, 0, 0 },
628 Gainsboro = { 0.864, 0.864, 0.864 },
629 GhostWhite = { 0.972, 0.972, 0.972 },
630 Gold = { 1, 0.844, 0.844 },
631 Goldenrod = { 0.855, 0.648, 0.648 },
632 Gray = { 0.5, 0.5, 0.5 },
633 Green = { 0, 0.5, 0.5 },
634 GreenYellow = { 0.68, 1, 1 },
635 Grey = { 0.5, 0.5, 0.5 },
636 Honeydew = { 0.94, 1, 1 },

```

```

637 HotPink = { 1, 0.41, 0.41 },
638 IndianRed = { 0.804, 0.36, 0.36 },
639 Indigo = { 0.294, 0, 0 },
640 Ivory = { 1, 1, 1 },
641 Khaki = { 0.94, 0.9, 0.9 },
642 Lavender = { 0.9, 0.9, 0.9 },
643 LavenderBlush = { 1, 0.94, 0.94 },
644 LawnGreen = { 0.488, 0.99, 0.99 },
645 LemonChiffon = { 1, 0.98, 0.98 },
646 LightBlue = { 0.68, 0.848, 0.848 },
647 LightCoral = { 0.94, 0.5, 0.5 },
648 LightCyan = { 0.88, 1, 1 },
649 LightGoldenrod = { 0.933, 0.867, 0.867 },
650 LightGoldenrodYellow = { 0.98, 0.98, 0.98 },
651 LightGray = { 0.828, 0.828, 0.828 },
652 LightGreen = { 0.565, 0.932, 0.932 },
653 LightGrey = { 0.828, 0.828, 0.828 },
654 LightPink = { 1, 0.712, 0.712 },
655 LightSalmon = { 1, 0.628, 0.628 },
656 LightSeaGreen = { 0.125, 0.698, 0.698 },
657 LightSkyBlue = { 0.53, 0.808, 0.808 },
658 LightSlateBlue = { 0.518, 0.44, 0.44 },
659 LightSlateGray = { 0.468, 0.532, 0.532 },
660 LightSlateGrey = { 0.468, 0.532, 0.532 },
661 LightSteelBlue = { 0.69, 0.77, 0.77 },
662 LightYellow = { 1, 1, 1 },
663 Lime = { 0, 1, 1 },
664 LimeGreen = { 0.196, 0.804, 0.804 },
665 Linen = { 0.98, 0.94, 0.94 },
666 Magenta = { 1, 0, 0 },
667 Maroon = { 0.5, 0, 0 },
668 MediumAquamarine = { 0.4, 0.804, 0.804 },
669 MediumBlue = { 0, 0, 0 },
670 MediumOrchid = { 0.73, 0.332, 0.332 },
671 MediumPurple = { 0.576, 0.44, 0.44 },
672 MediumSeaGreen = { 0.235, 0.7, 0.7 },
673 MediumSlateBlue = { 0.484, 0.408, 0.408 },
674 MediumSpringGreen = { 0, 0.98, 0.98 },
675 MediumTurquoise = { 0.284, 0.82, 0.82 },
676 MediumVioletRed = { 0.78, 0.084, 0.084 },
677 MidnightBlue = { 0.098, 0.098, 0.098 },
678 MintCream = { 0.96, 1, 1 },
679 MistyRose = { 1, 0.894, 0.894 },
680 Moccasin = { 1, 0.894, 0.894 },
681 NavajoWhite = { 1, 0.87, 0.87 },
682 Navy = { 0, 0, 0 },
683 NavyBlue = { 0, 0, 0 },
684 OldLace = { 0.992, 0.96, 0.96 },
685 Olive = { 0.5, 0.5, 0.5 },
686 OliveDrab = { 0.42, 0.556, 0.556 },
687 Orange = { 1, 0.648, 0.648 },
688 OrangeRed = { 1, 0.27, 0.27 },
689 Orchid = { 0.855, 0.44, 0.44 },
690 PaleGoldenrod = { 0.932, 0.91, 0.91 },
691 PaleGreen = { 0.596, 0.985, 0.985 },
692 PaleTurquoise = { 0.688, 0.932, 0.932 },
693 PaleVioletRed = { 0.86, 0.44, 0.44 },
694 PapayaWhip = { 1, 0.936, 0.936 },
695 PeachPuff = { 1, 0.855, 0.855 },
696 Peru = { 0.804, 0.52, 0.52 },
697 Pink = { 1, 0.752, 0.752 },
698 Plum = { 0.868, 0.628, 0.628 },
699 PowderBlue = { 0.69, 0.88, 0.88 },
700 Purple = { 0.5, 0, 0 },

```

```

701 Red = { 1, 0, 0 },
702 RosyBrown = { 0.736, 0.56, 0.56 },
703 RoyalBlue = { 0.255, 0.41, 0.41 },
704 SaddleBrown = { 0.545, 0.27, 0.27 },
705 Salmon = { 0.98, 0.5, 0.5 },
706 SandyBrown = { 0.956, 0.644, 0.644 },
707 SeaGreen = { 0.18, 0.545, 0.545 },
708 Seashell = { 1, 0.96, 0.96 },
709 Sienna = { 0.628, 0.32, 0.32 },
710 Silver = { 0.752, 0.752, 0.752 },
711 SkyBlue = { 0.53, 0.808, 0.808 },
712 SlateBlue = { 0.415, 0.352, 0.352 },
713 SlateGray = { 0.44, 0.5, 0.5 },
714 SlateGrey = { 0.44, 0.5, 0.5 },
715 Snow = { 1, 0.98, 0.98 },
716 SpringGreen = { 0, 1, 1 },
717 SteelBlue = { 0.275, 0.51, 0.51 },
718 Tan = { 0.824, 0.705, 0.705 },
719 Teal = { 0, 0.5, 0.5 },
720 Thistle = { 0.848, 0.75, 0.75 },
721 Tomato = { 1, 0.39, 0.39 },
722 Turquoise = { 0.25, 0.88, 0.88 },
723 Violet = { 0.932, 0.51, 0.51 },
724 VioletRed = { 0.816, 0.125, 0.125 },
725 Wheat = { 0.96, 0.87, 0.87 },
726 White = { 1, 1, 1 },
727 WhiteSmoke = { 0.96, 0.96, 0.96 },
728 Yellow = { 1, 1, 1 },
729 YellowGreen = { 0.604, 0.804, 0.804 },
730
731 ---x11
732 AntiqueWhite1 = { 1, 0.936, 0.936 },
733 AntiqueWhite2 = { 0.932, 0.875, 0.875 },
734 AntiqueWhite3 = { 0.804, 0.752, 0.752 },
735 AntiqueWhite4 = { 0.545, 0.512, 0.512 },
736 Aquamarine1 = { 0.498, 1, 1 },
737 Aquamarine2 = { 0.464, 0.932, 0.932 },
738 Aquamarine3 = { 0.4, 0.804, 0.804 },
739 Aquamarine4 = { 0.27, 0.545, 0.545 },
740 Azure1 = { 0.94, 1, 1 },
741 Azure2 = { 0.88, 0.932, 0.932 },
742 Azure3 = { 0.756, 0.804, 0.804 },
743 Azure4 = { 0.512, 0.545, 0.545 },
744 Bisque1 = { 1, 0.894, 0.894 },
745 Bisque2 = { 0.932, 0.835, 0.835 },
746 Bisque3 = { 0.804, 0.716, 0.716 },
747 Bisque4 = { 0.545, 0.49, 0.49 },
748 Blue1 = { 0, 0, 0 },
749 Blue2 = { 0, 0, 0 },
750 Blue3 = { 0, 0, 0 },
751 Blue4 = { 0, 0, 0 },
752 Brown1 = { 1, 0.25, 0.25 },
753 Brown2 = { 0.932, 0.23, 0.23 },
754 Brown3 = { 0.804, 0.2, 0.2 },
755 Brown4 = { 0.545, 0.136, 0.136 },
756 Burlywood1 = { 1, 0.828, 0.828 },
757 Burlywood2 = { 0.932, 0.772, 0.772 },
758 Burlywood3 = { 0.804, 0.668, 0.668 },
759 Burlywood4 = { 0.545, 0.45, 0.45 },
760 CadetBlue1 = { 0.596, 0.96, 0.96 },
761 CadetBlue2 = { 0.556, 0.898, 0.898 },
762 CadetBlue3 = { 0.48, 0.772, 0.772 },
763 CadetBlue4 = { 0.325, 0.525, 0.525 },
764 Chartreuse1 = { 0.498, 1, 1 },

```

```

765 Chartreuse2 = { 0.464, 0.932, 0.932 },
766 Chartreuse3 = { 0.4, 0.804, 0.804 },
767 Chartreuse4 = { 0.27, 0.545, 0.545 },
768 Chocolate1 = { 1, 0.498, 0.498 },
769 Chocolate2 = { 0.932, 0.464, 0.464 },
770 Chocolate3 = { 0.804, 0.4, 0.4 },
771 Chocolate4 = { 0.545, 0.27, 0.27 },
772 Coral1 = { 1, 0.448, 0.448 },
773 Coral2 = { 0.932, 0.415, 0.415 },
774 Coral3 = { 0.804, 0.356, 0.356 },
775 Coral4 = { 0.545, 0.244, 0.244 },
776 Cornsilk1 = { 1, 0.972, 0.972 },
777 Cornsilk2 = { 0.932, 0.91, 0.91 },
778 Cornsilk3 = { 0.804, 0.785, 0.785 },
779 Cornsilk4 = { 0.545, 0.532, 0.532 },
780 Cyan1 = { 0, 1, 1 },
781 Cyan2 = { 0, 0.932, 0.932 },
782 Cyan3 = { 0, 0.804, 0.804 },
783 Cyan4 = { 0, 0.545, 0.545 },
784 DarkGoldenrod1 = { 1, 0.725, 0.725 },
785 DarkGoldenrod2 = { 0.932, 0.68, 0.68 },
786 DarkGoldenrod3 = { 0.804, 0.585, 0.585 },
787 DarkGoldenrod4 = { 0.545, 0.396, 0.396 },
788 DarkOliveGreen1 = { 0.792, 1, 1 },
789 DarkOliveGreen2 = { 0.736, 0.932, 0.932 },
790 DarkOliveGreen3 = { 0.635, 0.804, 0.804 },
791 DarkOliveGreen4 = { 0.43, 0.545, 0.545 },
792 DarkOrange1 = { 1, 0.498, 0.498 },
793 DarkOrange2 = { 0.932, 0.464, 0.464 },
794 DarkOrange3 = { 0.804, 0.4, 0.4 },
795 DarkOrange4 = { 0.545, 0.27, 0.27 },
796 DarkOrchid1 = { 0.75, 0.244, 0.244 },
797 DarkOrchid2 = { 0.698, 0.228, 0.228 },
798 DarkOrchid3 = { 0.604, 0.196, 0.196 },
799 DarkOrchid4 = { 0.408, 0.132, 0.132 },
800 DarkSeaGreen1 = { 0.756, 1, 1 },
801 DarkSeaGreen2 = { 0.705, 0.932, 0.932 },
802 DarkSeaGreen3 = { 0.608, 0.804, 0.804 },
803 DarkSeaGreen4 = { 0.41, 0.545, 0.545 },
804 DarkSlateGray1 = { 0.592, 1, 1 },
805 DarkSlateGray2 = { 0.552, 0.932, 0.932 },
806 DarkSlateGray3 = { 0.475, 0.804, 0.804 },
807 DarkSlateGray4 = { 0.32, 0.545, 0.545 },
808 DeepPink1 = { 1, 0.08, 0.08 },
809 DeepPink2 = { 0.932, 0.07, 0.07 },
810 DeepPink3 = { 0.804, 0.064, 0.064 },
811 DeepPink4 = { 0.545, 0.04, 0.04 },
812 DeepSkyBlue1 = { 0, 0.75, 0.75 },
813 DeepSkyBlue2 = { 0, 0.698, 0.698 },
814 DeepSkyBlue3 = { 0, 0.604, 0.604 },
815 DeepSkyBlue4 = { 0, 0.408, 0.408 },
816 DodgerBlue1 = { 0.116, 0.565, 0.565 },
817 DodgerBlue2 = { 0.11, 0.525, 0.525 },
818 DodgerBlue3 = { 0.094, 0.455, 0.455 },
819 DodgerBlue4 = { 0.064, 0.305, 0.305 },
820 Firebrick1 = { 1, 0.19, 0.19 },
821 Firebrick2 = { 0.932, 0.172, 0.172 },
822 Firebrick3 = { 0.804, 0.15, 0.15 },
823 Firebrick4 = { 0.545, 0.1, 0.1 },
824 Gold1 = { 1, 0.844, 0.844 },
825 Gold2 = { 0.932, 0.79, 0.79 },
826 Gold3 = { 0.804, 0.68, 0.68 },
827 Gold4 = { 0.545, 0.46, 0.46 },
828 Goldenrod1 = { 1, 0.756, 0.756 },

```

```

829 Goldenrod2 = { 0.932, 0.705, 0.705 },
830 Goldenrod3 = { 0.804, 0.608, 0.608 },
831 Goldenrod4 = { 0.545, 0.41, 0.41 },
832 Green1 = { 0, 1, 1 },
833 Green2 = { 0, 0.932, 0.932 },
834 Green3 = { 0, 0.804, 0.804 },
835 Green4 = { 0, 0.545, 0.545 },
836 Honeydew1 = { 0.94, 1, 1 },
837 Honeydew2 = { 0.88, 0.932, 0.932 },
838 Honeydew3 = { 0.756, 0.804, 0.804 },
839 Honeydew4 = { 0.512, 0.545, 0.545 },
840 HotPink1 = { 1, 0.43, 0.43 },
841 HotPink2 = { 0.932, 0.415, 0.415 },
842 HotPink3 = { 0.804, 0.376, 0.376 },
843 HotPink4 = { 0.545, 0.228, 0.228 },
844 IndianRed1 = { 1, 0.415, 0.415 },
845 IndianRed2 = { 0.932, 0.39, 0.39 },
846 IndianRed3 = { 0.804, 0.332, 0.332 },
847 IndianRed4 = { 0.545, 0.228, 0.228 },
848 Ivory1 = { 1, 1, 1 },
849 Ivory2 = { 0.932, 0.932, 0.932 },
850 Ivory3 = { 0.804, 0.804, 0.804 },
851 Ivory4 = { 0.545, 0.545, 0.545 },
852 Khaki1 = { 1, 0.965, 0.965 },
853 Khaki2 = { 0.932, 0.9, 0.9 },
854 Khaki3 = { 0.804, 0.776, 0.776 },
855 Khaki4 = { 0.545, 0.525, 0.525 },
856 LavenderBlush1 = { 1, 0.94, 0.94 },
857 LavenderBlush2 = { 0.932, 0.88, 0.88 },
858 LavenderBlush3 = { 0.804, 0.756, 0.756 },
859 LavenderBlush4 = { 0.545, 0.512, 0.512 },
860 LemonChiffon1 = { 1, 0.98, 0.98 },
861 LemonChiffon2 = { 0.932, 0.912, 0.912 },
862 LemonChiffon3 = { 0.804, 0.79, 0.79 },
863 LemonChiffon4 = { 0.545, 0.536, 0.536 },
864 LightBlue1 = { 0.75, 0.936, 0.936 },
865 LightBlue2 = { 0.698, 0.875, 0.875 },
866 LightBlue3 = { 0.604, 0.752, 0.752 },
867 LightBlue4 = { 0.408, 0.512, 0.512 },
868 LightCyan1 = { 0.88, 1, 1 },
869 LightCyan2 = { 0.82, 0.932, 0.932 },
870 LightCyan3 = { 0.705, 0.804, 0.804 },
871 LightCyan4 = { 0.48, 0.545, 0.545 },
872 LightGoldenrod1 = { 1, 0.925, 0.925 },
873 LightGoldenrod2 = { 0.932, 0.864, 0.864 },
874 LightGoldenrod3 = { 0.804, 0.745, 0.745 },
875 LightGoldenrod4 = { 0.545, 0.505, 0.505 },
876 LightPink1 = { 1, 0.684, 0.684 },
877 LightPink2 = { 0.932, 0.635, 0.635 },
878 LightPink3 = { 0.804, 0.55, 0.55 },
879 LightPink4 = { 0.545, 0.372, 0.372 },
880 LightSalmon1 = { 1, 0.628, 0.628 },
881 LightSalmon2 = { 0.932, 0.585, 0.585 },
882 LightSalmon3 = { 0.804, 0.505, 0.505 },
883 LightSalmon4 = { 0.545, 0.34, 0.34 },
884 LightSkyBlue1 = { 0.69, 0.888, 0.888 },
885 LightSkyBlue2 = { 0.644, 0.828, 0.828 },
886 LightSkyBlue3 = { 0.552, 0.712, 0.712 },
887 LightSkyBlue4 = { 0.376, 0.484, 0.484 },
888 LightSteelBlue1 = { 0.792, 0.884, 0.884 },
889 LightSteelBlue2 = { 0.736, 0.824, 0.824 },
890 LightSteelBlue3 = { 0.635, 0.71, 0.71 },
891 LightSteelBlue4 = { 0.43, 0.484, 0.484 },
892 LightYellow1 = { 1, 1, 1 },

```



```

893 LightYellow2 = { 0.932, 0.932, 0.932 },
894 LightYellow3 = { 0.804, 0.804, 0.804 },
895 LightYellow4 = { 0.545, 0.545, 0.545 },
896 Magenta1 = { 1, 0, 0 },
897 Magenta2 = { 0.932, 0, 0 },
898 Magenta3 = { 0.804, 0, 0 },
899 Magenta4 = { 0.545, 0, 0 },
900 Maroon1 = { 1, 0.204, 0.204 },
901 Maroon2 = { 0.932, 0.19, 0.19 },
902 Maroon3 = { 0.804, 0.16, 0.16 },
903 Maroon4 = { 0.545, 0.11, 0.11 },
904 MediumOrchid1 = { 0.88, 0.4, 0.4 },
905 MediumOrchid2 = { 0.82, 0.372, 0.372 },
906 MediumOrchid3 = { 0.705, 0.32, 0.32 },
907 MediumOrchid4 = { 0.48, 0.215, 0.215 },
908 MediumPurple1 = { 0.67, 0.51, 0.51 },
909 MediumPurple2 = { 0.624, 0.475, 0.475 },
910 MediumPurple3 = { 0.536, 0.408, 0.408 },
911 MediumPurple4 = { 0.365, 0.28, 0.28 },
912 MistyRose1 = { 1, 0.894, 0.894 },
913 MistyRose2 = { 0.932, 0.835, 0.835 },
914 MistyRose3 = { 0.804, 0.716, 0.716 },
915 MistyRose4 = { 0.545, 0.49, 0.49 },
916 NavajoWhite1 = { 1, 0.87, 0.87 },
917 NavajoWhite2 = { 0.932, 0.81, 0.81 },
918 NavajoWhite3 = { 0.804, 0.7, 0.7 },
919 NavajoWhite4 = { 0.545, 0.475, 0.475 },
920 OliveDrab1 = { 0.752, 1, 1 },
921 OliveDrab2 = { 0.7, 0.932, 0.932 },
922 OliveDrab3 = { 0.604, 0.804, 0.804 },
923 OliveDrab4 = { 0.41, 0.545, 0.545 },
924 Orange1 = { 1, 0.648, 0.648 },
925 Orange2 = { 0.932, 0.604, 0.604 },
926 Orange3 = { 0.804, 0.52, 0.52 },
927 Orange4 = { 0.545, 0.352, 0.352 },
928 OrangeRed1 = { 1, 0.27, 0.27 },
929 OrangeRed2 = { 0.932, 0.25, 0.25 },
930 OrangeRed3 = { 0.804, 0.215, 0.215 },
931 OrangeRed4 = { 0.545, 0.145, 0.145 },
932 Orchid1 = { 1, 0.512, 0.512 },
933 Orchid2 = { 0.932, 0.48, 0.48 },
934 Orchid3 = { 0.804, 0.41, 0.41 },
935 Orchid4 = { 0.545, 0.28, 0.28 },
936 PaleGreen1 = { 0.604, 1, 1 },
937 PaleGreen2 = { 0.565, 0.932, 0.932 },
938 PaleGreen3 = { 0.488, 0.804, 0.804 },
939 PaleGreen4 = { 0.33, 0.545, 0.545 },
940 PaleTurquoise1 = { 0.732, 1, 1 },
941 PaleTurquoise2 = { 0.684, 0.932, 0.932 },
942 PaleTurquoise3 = { 0.59, 0.804, 0.804 },
943 PaleTurquoise4 = { 0.4, 0.545, 0.545 },
944 PaleVioletRed1 = { 1, 0.51, 0.51 },
945 PaleVioletRed2 = { 0.932, 0.475, 0.475 },
946 PaleVioletRed3 = { 0.804, 0.408, 0.408 },
947 PaleVioletRed4 = { 0.545, 0.28, 0.28 },
948 PeachPuff1 = { 1, 0.855, 0.855 },
949 PeachPuff2 = { 0.932, 0.796, 0.796 },
950 PeachPuff3 = { 0.804, 0.688, 0.688 },
951 PeachPuff4 = { 0.545, 0.468, 0.468 },
952 Pink1 = { 1, 0.71, 0.71 },
953 Pink2 = { 0.932, 0.664, 0.664 },
954 Pink3 = { 0.804, 0.57, 0.57 },
955 Pink4 = { 0.545, 0.39, 0.39 },
956 Plum1 = { 1, 0.732, 0.732 },

```

```

957 Plum2 = { 0.932, 0.684, 0.684 },
958 Plum3 = { 0.804, 0.59, 0.59 },
959 Plum4 = { 0.545, 0.4, 0.4 },
960 Purple1 = { 0.608, 0.19, 0.19 },
961 Purple2 = { 0.57, 0.172, 0.172 },
962 Purple3 = { 0.49, 0.15, 0.15 },
963 Purple4 = { 0.332, 0.1, 0.1 },
964 Red1 = { 1, 0, 0 },
965 Red2 = { 0.932, 0, 0 },
966 Red3 = { 0.804, 0, 0 },
967 Red4 = { 0.545, 0, 0 },
968 RosyBrown1 = { 1, 0.756, 0.756 },
969 RosyBrown2 = { 0.932, 0.705, 0.705 },
970 RosyBrown3 = { 0.804, 0.608, 0.608 },
971 RosyBrown4 = { 0.545, 0.41, 0.41 },
972 RoyalBlue1 = { 0.284, 0.464, 0.464 },
973 RoyalBlue2 = { 0.264, 0.43, 0.43 },
974 RoyalBlue3 = { 0.228, 0.372, 0.372 },
975 RoyalBlue4 = { 0.152, 0.25, 0.25 },
976 Salmon1 = { 1, 0.55, 0.55 },
977 Salmon2 = { 0.932, 0.51, 0.51 },
978 Salmon3 = { 0.804, 0.44, 0.44 },
979 Salmon4 = { 0.545, 0.298, 0.298 },
980 SeaGreen1 = { 0.33, 1, 1 },
981 SeaGreen2 = { 0.305, 0.932, 0.932 },
982 SeaGreen3 = { 0.264, 0.804, 0.804 },
983 SeaGreen4 = { 0.18, 0.545, 0.545 },
984 Seashell1 = { 1, 0.96, 0.96 },
985 Seashell2 = { 0.932, 0.898, 0.898 },
986 Seashell3 = { 0.804, 0.772, 0.772 },
987 Seashell4 = { 0.545, 0.525, 0.525 },
988 Sienna1 = { 1, 0.51, 0.51 },
989 Sienna2 = { 0.932, 0.475, 0.475 },
990 Sienna3 = { 0.804, 0.408, 0.408 },
991 Sienna4 = { 0.545, 0.28, 0.28 },
992 SkyBlue1 = { 0.53, 0.808, 0.808 },
993 SkyBlue2 = { 0.494, 0.752, 0.752 },
994 SkyBlue3 = { 0.424, 0.65, 0.65 },
995 SkyBlue4 = { 0.29, 0.44, 0.44 },
996 SlateBlue1 = { 0.512, 0.435, 0.435 },
997 SlateBlue2 = { 0.48, 0.404, 0.404 },
998 SlateBlue3 = { 0.41, 0.35, 0.35 },
999 SlateBlue4 = { 0.28, 0.235, 0.235 },
1000 SlateGray1 = { 0.776, 0.888, 0.888 },
1001 SlateGray2 = { 0.725, 0.828, 0.828 },
1002 SlateGray3 = { 0.624, 0.712, 0.712 },
1003 SlateGray4 = { 0.424, 0.484, 0.484 },
1004 Snow1 = { 1, 0.98, 0.98 },
1005 Snow2 = { 0.932, 0.912, 0.912 },
1006 Snow3 = { 0.804, 0.79, 0.79 },
1007 Snow4 = { 0.545, 0.536, 0.536 },
1008 SpringGreen1 = { 0, 1, 1 },
1009 SpringGreen2 = { 0, 0.932, 0.932 },
1010 SpringGreen3 = { 0, 0.804, 0.804 },
1011 SpringGreen4 = { 0, 0.545, 0.545 },
1012 SteelBlue1 = { 0.39, 0.72, 0.72 },
1013 SteelBlue2 = { 0.36, 0.675, 0.675 },
1014 SteelBlue3 = { 0.31, 0.58, 0.58 },
1015 SteelBlue4 = { 0.21, 0.392, 0.392 },
1016 Tan1 = { 1, 0.648, 0.648 },
1017 Tan2 = { 0.932, 0.604, 0.604 },
1018 Tan3 = { 0.804, 0.52, 0.52 },
1019 Tan4 = { 0.545, 0.352, 0.352 },
1020 Thistle1 = { 1, 0.884, 0.884 },

```

```

1021 Thistle2 = { 0.932, 0.824, 0.824 },
1022 Thistle3 = { 0.804, 0.71, 0.71 },
1023 Thistle4 = { 0.545, 0.484, 0.484 },
1024 Tomato1 = { 1, 0.39, 0.39 },
1025 Tomato2 = { 0.932, 0.36, 0.36 },
1026 Tomato3 = { 0.804, 0.31, 0.31 },
1027 Tomato4 = { 0.545, 0.21, 0.21 },
1028 Turquoise1 = { 0, 0.96, 0.96 },
1029 Turquoise2 = { 0, 0.898, 0.898 },
1030 Turquoise3 = { 0, 0.772, 0.772 },
1031 Turquoise4 = { 0, 0.525, 0.525 },
1032 VioletRed1 = { 1, 0.244, 0.244 },
1033 VioletRed2 = { 0.932, 0.228, 0.228 },
1034 VioletRed3 = { 0.804, 0.196, 0.196 },
1035 VioletRed4 = { 0.545, 0.132, 0.132 },
1036 Wheat1 = { 1, 0.905, 0.905 },
1037 Wheat2 = { 0.932, 0.848, 0.848 },
1038 Wheat3 = { 0.804, 0.73, 0.73 },
1039 Wheat4 = { 0.545, 0.494, 0.494 },
1040 Yellow1 = { 1, 1, 1 },
1041 Yellow2 = { 0.932, 0.932, 0.932 },
1042 Yellow3 = { 0.804, 0.804, 0.804 },
1043 Yellow4 = { 0.545, 0.545, 0.545 },
1044 Gray0 = { 0.745, 0.745, 0.745 },
1045 Green0 = { 0, 1, 1 },
1046 Grey0 = { 0.745, 0.745, 0.745 },
1047 Maroon0 = { 0.69, 0.19, 0.19 },
1048 Purple0 = { 0.628, 0.125, 0.125 },
1049 }
1050
1051 --- https://luarocks.org/modules/Firanel/lua-color
1052 --- Copyright (c) 2021 Firanel
1053
1054 ---https://github.com/Firanel/lua-color/blob/master/util/bitwise.lua
1055 local bitwise = (function()
1056   -- Implementations of bitwise operators so that lua-color can be used
1057   -- with Lua 5.1 and LuaJIT 2.1.0-beta3 (e.g. inside Neovim).
1058
1059   -- Code taken directly from:
1060   --
1061   --> https://stackoverflow.com/questions/5977654/how-do-i-use-the-bitwise-operator-xor-in-lua
1062
1063   local function bit_xor(a, b)
1064     local p, c = 1, 0
1065     while a > 0 and b > 0 do
1066       local ra, rb = a % 2, b % 2
1067       if ra ~= rb then
1068         c = c + p
1069       end
1070       a, b, p = (a - ra) / 2, (b - rb) / 2, p * 2
1071     end
1072     if a < b then
1073       a = b
1074     end
1075     while a > 0 do
1076       local ra = a % 2
1077       if ra > 0 then
1078         c = c + p
1079       end
1080       a, p = (a - ra) / 2, p * 2
1081     end
1082     return c
1083   end

```

```

1084 local function bit_or(a, b)
1085     local p, c = 1, 0
1086     while a + b > 0 do
1087         local ra, rb = a % 2, b % 2
1088         if ra + rb > 0 then
1089             c = c + p
1090         end
1091         a, b, p = (a - ra) / 2, (b - rb) / 2, p * 2
1092     end
1093     return c
1094 end
1095
1096 local function bit_not(n)
1097     local p, c = 1, 0
1098     while n > 0 do
1099         local r = n % 2
1100         if r < 1 then
1101             c = c + p
1102         end
1103         n, p = (n - r) / 2, p * 2
1104     end
1105     return c
1106 end
1107
1108 local function bit_and(a, b)
1109     local p, c = 1, 0
1110     while a > 0 and b > 0 do
1111         local ra, rb = a % 2, b % 2
1112         if ra + rb > 1 then
1113             c = c + p
1114         end
1115         a, b, p = (a - ra) / 2, (b - rb) / 2, p * 2
1116     end
1117     return c
1118 end
1119
1120 local function bit_lshift(x, by)
1121     return x * 2 ^ by
1122 end
1123
1124 local function bit_rshift(x, by)
1125     return math.floor(x / 2 ^ by)
1126 end
1127
1128 return {
1129     bit_xor = bit_xor,
1130     bit_or = bit_or,
1131     bit_not = bit_not,
1132     bit_and = bit_and,
1133     bit_lshift = bit_lshift,
1134     bit_rshift = bit_rshift,
1135 }
1136 end()
1137
1138 --- https://github.com/Firanel/lua-color/blob/master/util/class.lua
1139 local class = (function()
1140
1141     -- Code based on:
1142     -- http://lua-users.org/wiki/SimpleLuaClasses
1143
1144     ---Helper function to create classes
1145     ---
1146     ---@usage local Color = class(function () --[[ constructor ]] end)
1147     ---@usage local Color2 = class(

```

```

1148 -- Color,
1149 -- function () --[[ constructor ]] end,
1150 -- { prop_a = "some value" }
1151 -- )
1152 local function class(base, init, defaults)
1153     local c = defaults or {} -- a new class instance
1154     if not init and type(base) == 'function' then
1155         init = base
1156         base = nil
1157     elseif type(base) == 'table' then
1158         -- our new class is a shallow copy of the base class!
1159         for i, v in pairs(base) do
1160             c[i] = v
1161         end
1162         c._base = base
1163     end
1164     -- the class will be the metatable for all its objects,
1165     -- and they will look up their methods in it.
1166     c.__index = c
1167
1168     -- expose a constructor which can be called by <classname>(<args>)
1169     local mt = {}
1170     mt.__call = function(class_tbl, ...)
1171         local obj = {}
1172         setmetatable(obj, c)
1173         if init then
1174             init(obj, ...)
1175         else
1176             -- make sure that any stuff from the base class is initialized!
1177             if base and base.init then
1178                 base.init(obj, ...)
1179             end
1180         end
1181         return obj
1182     end
1183     c.init = init
1184     c.is_a = function(self, klass)
1185         local m = getmetatable(self)
1186         while m do
1187             if m == klass then
1188                 return true
1189             end
1190             m = m._base
1191         end
1192         return false
1193     end
1194     setmetatable(c, mt)
1195     return c
1196 end
1197
1198 return class
1199 end()
1200
1201 --https://github.com/Firanel/luas-color/blob/master/luas/init.lua
1202 local utils = (function()
1203     local function min_ind(first, ...)
1204         local min, ind = first, 1
1205         for i, v in ipairs { ... } do
1206             if v < min then
1207                 min, ind = v, i + 1
1208             end
1209         end
1210         return min, ind
1211     end

```

```

1212
1213 local function max_ind(first, ...)
1214     local max, ind = first, 1
1215     for i, v in ipairs { ... } do
1216         if v > max then
1217             max, ind = v, i + 1
1218         end
1219     end
1220     return max, ind
1221 end
1222
1223 local function round(x)
1224     return x + 0.5 - (x + 0.5) % 1
1225 end
1226
1227 local function clamp(x, min, max)
1228     return x < min and min or x > max and max or x
1229 end
1230
1231 local function map(t, cb)
1232     local n = {}
1233     for i, v in ipairs(t) do
1234         n[i] = cb(v)
1235     end
1236     return n
1237 end
1238
1239 return {
1240     min = min_ind,
1241     max = max_ind,
1242     round = round,
1243     clamp = clamp,
1244     map = map,
1245 }
1246 end()
1247
1248 ---Source:
1249 ⇨ [texmf-dist/tex/context/base/mkiv/attr-col.lua](https://git.texlive.info/texlive/tree/Master/texmf-dist/
1249 local convert = (function()
1250
1251     ---
1252     ---https://www.rapidtables.com/convert/color/rgb-to-cmyk.html
1253     ---https://www.101computing.net/cmyk-to-rgb-conversion-algorithm/
1254     ---
1255     ---@param r r # red (0.0 - 1.0)
1256     ---@param g g # green (0.0 - 1.0)
1257     ---@param b b # blue (0.0 - 1.0)
1258     ---
1259     ---@return c c # cyan (0.0 - 1.0)
1260     ---@return m m # magenta (0.0 - 1.0)
1261     ---@return y y # yellow (0.0 - 1.0)
1262     ---@return k k # key(black) (0.0 - 1.0)
1263     local function rgb_to_cmyk(r, g, b)
1264         local K = math.max(r, g, b)
1265         if K == 0 then
1266             return 0.0, 0.0, 0.0, 1.0
1267         end
1268         local k = 1 - K
1269         local c = (K - r) / K
1270         local m = (K - g) / K
1271         local y = (K - b) / K
1272         return c, m, y, k
1273     end
1274

```

```

1275  ---https://www.rapidtables.com/convert/color/cmyk-to-rgb.html
1276  local function cmyk_to_rgb(c, m, y, k)
1277      if not k then
1278          k = 0
1279      end
1280      ---te\mf-dist\tex\context\base\mkiv\attr-col.lua
1281      -- local d = 1.0 - k
1282      -- local r = 1.0 - math.min(1.0, c * d + k)
1283      -- local g = 1.0 - math.min(1.0, m * d + k)
1284      -- local b = 1.0 - math.min(1.0, y * d + k)
1285
1286      ---te\mf-dist\tex\context\base\mkiv\attr-col.lua
1287      -- local r = 1.0 - math.min(1.0, c + k)
1288      -- local g = 1.0 - math.min(1.0, m + k)
1289      -- local b = 1.0 - math.min(1.0, y + k)
1290
1291
1292      ↪ ---https://github.com/Firanel/lua-color/blob/eba73e53e9abd2e8da4d56b016fd77b45c2f3b79/init.lua#L335
1293      local K = 1 - k
1294      local r = (1 - c) * K
1295      local g = (1 - m) * K
1296      local b = (1 - y) * K
1297
1298      return r, g, b
1299  end
1300
1301  local function rgb_to_gray(r, g, b)
1302      if not r then
1303          return 0
1304      end
1305      local w = colors.weightgray
1306      if w == true then
1307          return .30 * r + .59 * g + .11 * b
1308      elseif not w then
1309          return r / 3 + g / 3 + b / 3
1310      else
1311          return w[1] * r + w[2] * g + w[3] * b
1312      end
1313  end
1314
1315  local function cmyk_to_gray(c, m, y, k)
1316      return rgb_to_gray(cmyk_to_rgb(c, m, y, k))
1317  end
1318
1319  -- http://en.wikipedia.org/wiki/HSI\_color\_space
1320  -- http://nl.wikipedia.org/wiki/HSV\_\(kleurruimte\)
1321
1322  --      h /= 60;          // sector 0 to 5
1323  --      i = floor( h );
1324  --      f = h - i;      // factorial part of h
1325
1326  local function hsv_to_rgb(h, s, v)
1327      if s > 1 then
1328          s = 1
1329      elseif s < 0 then
1330          s = 0
1331      elseif s == 0 then
1332          return v, v, v
1333      end
1334      if v > 1 then
1335          v = 1
1336      elseif v < 0 then
1337          v = 0

```

```

1338     end
1339     if h < 0 then
1340         h = 0
1341     elseif h >= 360 then
1342         h = mod(h, 360)
1343     end
1344     local hd = h / 60
1345     local hi = floor(hd)
1346     local f = hd - hi
1347     local p = v * (1 - s)
1348     local q = v * (1 - f * s)
1349     local t = v * (1 - (1 - f) * s)
1350     if hi == 0 then
1351         return v, t, p
1352     elseif hi == 1 then
1353         return q, v, p
1354     elseif hi == 2 then
1355         return p, v, t
1356     elseif hi == 3 then
1357         return p, q, v
1358     elseif hi == 4 then
1359         return t, p, v
1360     elseif hi == 5 then
1361         return v, p, q
1362     else
1363         print('error in hsv -> rgb', h, s, v)
1364         return 0, 0, 0
1365     end
1366 end
1367
1368 local function rgb_to_hsv(r, g, b)
1369     local offset, maximum, other_1, other_2
1370     if r >= g and r >= b then
1371         offset, maximum, other_1, other_2 = 0, r, g, b
1372     elseif g >= r and g >= b then
1373         offset, maximum, other_1, other_2 = 2, g, b, r
1374     else
1375         offset, maximum, other_1, other_2 = 4, b, r, g
1376     end
1377     if maximum == 0 then
1378         return 0, 0, 0
1379     end
1380     local minimum = other_1 < other_2 and other_1 or other_2
1381     if maximum == minimum then
1382         return 0, 0, maximum
1383     end
1384     local delta = maximum - minimum
1385     return (offset + (other_1 - other_2) / delta) * 60, delta / maximum,
1386         maximum
1387 end
1388
1389 local function gray_to_rgb(s) -- unweighted
1390     return 1 - s, 1 - s, 1 - s
1391 end
1392
1393 local function hsv_to_gray(h, s, v)
1394     return rgb_to_gray(hsv_to_rgb(h, s, v))
1395 end
1396
1397 local function gray_to_hsv(s)
1398     return 0, 0, s
1399 end
1400
1401 ---

```



```

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

```

```

1466     r, g, b = c, x, 0
1467 elseif h <= 2 then
1468     r, g, b = x, c, 0
1469 elseif h <= 3 then
1470     r, g, b = 0, c, x
1471 elseif h <= 4 then
1472     r, g, b = 0, x, c
1473 elseif h <= 5 then
1474     r, g, b = x, 0, c
1475 elseif h <= 6 then
1476     r, g, b = c, 0, x
1477 end
1478
1479 return r + m, g + m, b + m
1480 end
1481
1482 local function tonumPercent(str)
1483     if str:sub(-1) == '%' then
1484         return tonumber(str:sub(1, #str - 1)) / 100
1485     end
1486     return tonumber(str)
1487 end
1488
1489 -- Color
1490
1491 ---Color constructor.
1492 ---
1493 -- @function Color:__call
1494 ---
1495 ---@param ?string/table/Color value Color value (default: `nil`)
1496 ---
1497 ---@see Color:set
1498
1499 ---Red component.
1500 -- @field r
1501
1502 ---Green component.
1503 -- @field g
1504
1505 ---Blue component.
1506 -- @field b
1507
1508 ---Alpha component.
1509 -- @field a
1510
1511 ---Color class
1512 local Color = class(nil, function(this, value)
1513     if value then
1514         if type(value) == 'string' then
1515             -- # gets expanded to ##
1516             value = string.gsub(value, '^##', '#')
1517         end
1518         this:set(value)
1519     end
1520 end, { __is_color = true, r = 0, g = 0, b = 0, a = 1 })
1521
1522 ---Clone color
1523 ---
1524 ---@return Color copy
1525 function Color:clone()
1526     return Color(self)
1527 end
1528
1529 ---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>`huba(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 ,.%%%]+)'
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]+%%?) [
1665             ↪ ,]+([x.%x]+%%?)'
1666     assert(h and s and l and a)
1667     return self:set{
1668         h = tonumber(h) / 360,
1669         s = tonumPercent(s),
1670         l = tonumPercent(l),
1671         a = tonumPercent(a),
1672     }
1673 elseif func == 'hwb' then
1674     local h, w, b =
1675         values:match '([x.%x]+) [ ,]+([x.%x]+%%?) [ ,]+([x.%x]+%%?)'
1676     assert(h and w and b)
1677     return self:set{
1678         h = tonumber(h) / 360,
1679         w = tonumPercent(w),
1680         b = tonumPercent(b),
1681     }
1682 elseif func == 'hwba' then
1683     local h, w, b, a =
1684         values:match '([x.%x]+) [ ,]+([x.%x]+%%?) [ ,]+([x.%x]+%%?) [
1685             ↪ ,]+([x.%x]+%%?)'
1686     assert(h and w and b and a)
1687     return self:set{
1688         h = tonumber(h) / 360,
1689         w = tonumPercent(w),
1690         b = tonumPercent(b),
1691         a = tonumPercent(a),
1692     }
1693 elseif func == 'cmyk' then
1694     local c, m, y, k =
1695         values:match '([x.%x]+%%?) [ ,]+([x.%x]+%%?) [ ,]+([x.%x]+%%?) [
1696             ↪ ,]+([x.%x]+%%?)'
1697     assert(c and m and y and k)
1698     return self:set{
1699         c = tonumPercent(c),
1700         m = tonumPercent(m),
1701         y = tonumPercent(y),
1702         k = tonumPercent(k),
1703     }
1704 end
1705 else
1706     local col, dist, w, b, a =
1707         value:match '([RGBCMYrgbcmy])(%d*) [ ,]+([x.%x]+%%?) [ ,]+([x.%x]+%%?) [
1708             ↪ ,]+([x.%x]+%%?)'
1709     if col == nil then
1710         col, dist, w, b, a =
1711             value:match '([RGBCMYrgbcmy])(%d*) [ ,]+([x.%x]+%%?) [ ,]+([x.%x]+%%?)'
1712     end
1713     if col then
1714         col = col:lower()
1715
1716         local h
1717         if col == 'r' then

```

```

1714         h = 0
1715     elseif col == 'y' then
1716         h = 1 / 6
1717     elseif col == 'g' then
1718         h = 2 / 6
1719     elseif col == 'c' then
1720         h = 3 / 6
1721     elseif col == 'b' then
1722         h = 4 / 6
1723     elseif col == 'm' then
1724         h = 5 / 6
1725     end
1726
1727     if #dist > 0 then
1728         h = h + tonumber(dist) / 600
1729     end
1730
1731     return self:set{
1732         h = h,
1733         w = tonumPercent(w),
1734         b = tonumPercent(b),
1735         a = a and tonumPercent(a) or 1,
1736     }
1737 end
1738 end
1739 else
1740     value = value:sub(2)
1741 end
1742
1743 local pattern
1744 local div = 0xff
1745 if #value == 3 then
1746     pattern = '(%x)(%x)(%x)'
1747     div = 0xf
1748 elseif #value == 4 then
1749     pattern = '(%x)(%x)(%x)(%x)'
1750     div = 0xf
1751 elseif #value == 6 then
1752     pattern = '(%x%x)(%x%x)(%x%x)'
1753 elseif #value == 8 then
1754     pattern = '(%x%x)(%x%x)(%x%x)(%x%x)'
1755 else
1756     error('Not a valid color: ' .. tostring(value))
1757 end
1758 local r, g, b, a = value:match(pattern)
1759 assert(r ~= nil, 'Not a valid color: ' .. tostring(value))
1760 self.r = tonumber(r, 16) / div
1761 self.g = tonumber(g, 16) / div
1762 self.b = tonumber(b, 16) / div
1763 self.a = a ~= nil and tonumber(a, 16) / div.a or 1
1764
1765 -- table with rgb
1766 elseif value[1] ~= nil then
1767     self.r = value[1]
1768     self.g = value[2]
1769     self.b = value[3]
1770     self.a = value[4] or self.a or 1
1771 elseif value.r ~= nil then
1772     self.r = value.r
1773     self.g = value.g or self.g
1774     self.b = value.b or self.b
1775     self.a = value.a or self.a
1776
1777 elseif value.c ~= nil then

```

```

1778     self.r, self.g, self.b = convert.cmyk_to_rgb(value.c, value.m,
1779         value.y, value.k)
1780     self.a = 1
1781
1782     -- table with hs[vl]
1783     elseif value.h ~= nil then
1784         if value.w ~= nil then -- hwb
1785             value.v = 1 - value.b
1786             value.s = 1 - value.w / value.v
1787         end
1788
1789         local hue, saturation = value.h, value.s
1790         assert(hue ~= nil, saturation ~= nil)
1791
1792         local r, g, b = 0, 0, 0
1793
1794         if value.v ~= nil then
1795             local v = value.v
1796             local chroma = saturation * v
1797             r, g, b = hcm_to_rgb(hue, chroma, v - chroma)
1798
1799         elseif value.l ~= nil then
1800             local lightness = value.l
1801             local chroma = (1 - math.abs(2 * lightness - 1)) * saturation
1802             r, g, b = hcm_to_rgb(hue, chroma, lightness - chroma / 2)
1803         end
1804
1805         self.r = r
1806         self.g = g
1807         self.b = b
1808         self.a = value.a or self.a or 1
1809
1810     else -- Single set mode
1811         if value.red then
1812             self.r = value.red
1813         end
1814         if value.green then
1815             self.g = value.green
1816         end
1817         if value.blue then
1818             self.b = value.blue
1819         end
1820         if value.alpha then
1821             self.a = value.alpha
1822         end
1823
1824         if value.lightness then
1825             local h, s, l = self:hsl()
1826             self:set{
1827                 h = value.hue or h,
1828                 s = value.saturation or s,
1829                 l = value.lightness or l,
1830             }
1831             value.hue = nil
1832             value.saturation = nil
1833         end
1834
1835         if value.whiteness or value.blackness then
1836             local h, w, b = self:hwb()
1837             self:set{
1838                 h = value.hue or h,
1839                 w = value.whiteness or w,
1840                 b = value.blackness or b,
1841             }

```

```

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:cmymk()
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

```



```

1906     hue = (4 + (r - g) / chroma) / 6
1907     end
1908
1909     local saturation = max == 0 and 0 or chroma / max
1910
1911     return hue, saturation, max, min
1912 end
1913
1914 ---Get hsv values.
1915 ---
1916 ---@return number[0;1] hue
1917 ---@return number[0;1] saturation
1918 ---@return number[0;1] value
1919 function Color:hsv()
1920     local h, s, v = self:_hsvm()
1921     return h, s, v
1922 end
1923
1924 ---Get hsv values.
1925 ---
1926 ---@return number[0;1] hue
1927 ---@return number[0;1] saturation
1928 ---@return number[0;1] value
1929 ---@return number[0;1] alpha
1930 function Color:hsva()
1931     local h, s, v = self:_hsvm()
1932     return h, s, v, self.a
1933 end
1934
1935 ---Get hsl values.
1936 ---
1937 ---@return number[0;1] hue
1938 ---@return number[0;1] saturation
1939 ---@return number[0;1] lightness
1940 function Color:hsl()
1941     local hue, _, max, min = self:_hsvm()
1942     local lightness = (max + min) / 2
1943
1944     local saturation = lightness == 0 and 0 or (max - lightness) /
1945         math.min(lightness, 1 - lightness)
1946
1947     if saturation ~= saturation then
1948         saturation = 0
1949     end
1950
1951     return hue, saturation, lightness
1952 end
1953
1954 ---Get hsl values.
1955 ---
1956 ---@return number[0;1] hue
1957 ---@return number[0;1] saturation
1958 ---@return number[0;1] lightness
1959 ---@return number[0;1] alpha
1960 function Color:hsla()
1961     local h, s, l = self:hsl()
1962     return h, s, l, self.a
1963 end
1964
1965 ---Get hwb values.
1966 ---
1967 ---@return number[0;1] hue
1968 ---@return number[0;1] whiteness
1969 ---@return number[0;1] blackness

```

```

1970 function Color:hw b()
1971     local h, s, v = self:hs v()
1972     local w = (1 - s) * v
1973     local b = 1 - v
1974     return h, w, b
1975 end
1976
1977 ---Get hwb values.
1978 ---
1979 ---@return number[0;1] hue
1980 ---@return number[0;1] whiteness
1981 ---@return number[0;1] blackness
1982 ---@return number[0;1] alpha
1983 function Color:hwba()
1984     local h, w, b = self:hw b()
1985     return h, w, b, self.a
1986 end
1987
1988 ---Get cmyk values.
1989 ---
1990 ---@return number[0;1] cyan
1991 ---@return number[0;1] magenta
1992 ---@return number[0;1] yellow
1993 ---@return number[0;1] key
1994 function Color:cm yk()
1995     return convert.rgb_to_cm yk(self.r, self.g, self.b)
1996 end
1997
1998 ---Rotate hue of color.
1999 ---
2000 ---@param number[0;1]|table value Part of full turn or table containing degree or
    ↪ radians
2001 ---
2002 ---@return Color self
2003 ---
2004 ---@usage color:rotate(0.5)
2005 ---@usage color:rotate {deg=180}
2006 ---@usage color:rotate {rad=math.pi}
2007 function Color:rotate(value)
2008     local r
2009     if type(value) == 'number' then
2010         r = value
2011     elseif value.rad ~= nil then
2012         r = value.rad / (math.pi * 2)
2013     elseif value.deg ~= nil then
2014         r = value.deg / 360
2015     else
2016         error('No valid argument')
2017     end
2018
2019     local h, s, v = self:hs v()
2020     h = (h + r) % 1
2021     self:set{ h = h, s = s, v = v, a = self.a }
2022
2023     return self
2024 end
2025
2026 ---Invert the color.
2027 ---
2028 ---@return Color self
2029 function Color:invert()
2030     self.r = 1 - self.r
2031     self.g = 1 - self.g
2032     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

```

```

2097     ---@return Color
2098     ---@return Color
2099     function Color:triad()
2100         local h, s, v = self:hsv()
2101         return self,
2102             Color { h = (h + 1 / 3) % 1, s = s, v = v, a = self.a },
2103             Color { h = (h + 2 / 3) % 1, s = s, v = v, a = self.a }
2104     end
2105
2106     ---Generate tetradic color scheme.
2107     ---
2108     ---@return Color self
2109     ---@return Color
2110     ---@return Color
2111     ---@return Color
2112     function Color:tetrad()
2113         local h, s, v = self:hsv()
2114         return self,
2115             Color { h = (h + 1 / 4) % 1, s = s, v = v, a = self.a },
2116             Color { h = (h + 2 / 4) % 1, s = s, v = v, a = self.a },
2117             Color { h = (h + 3 / 4) % 1, s = s, v = v, a = self.a }
2118     end
2119
2120     ---Generate compound color scheme.
2121     ---
2122     ---@return Color
2123     ---@return Color self
2124     ---@return Color
2125     function Color:compound()
2126         local ca, _, cb = self:complement():analogous()
2127         return ca, self, cb
2128     end
2129
2130     ---Generate evenly spaced color scheme.
2131     -- <br>
2132     -- Generalization of `triad` and `tetrad`.
2133     ---
2134     ---@param int     n Return n colors
2135     ---@param ?number r Space colors over r rotations (Default: 1)
2136     ---
2137     ---@return {Color,...} Table with n colors including self at index 1
2138     function Color:evenlySpaced(n, r)
2139         assert(n > 0, 'n needs to be greater than 0')
2140         r = r or 1
2141
2142         local res = { self }
2143
2144         local rot = r / n
2145         local h, s, v = self:hsv()
2146         local a = self.a
2147
2148         for i = 1, n - 1 do
2149             h = (h + rot) % 1
2150             table.insert(res, Color { h = h, s = s, v = v, a = a })
2151         end
2152
2153         return res
2154     end
2155
2156     ---Get string representation of color.
2157     ---
2158     -- If `format` is `nil`, `color:tostring()` is the same as `tostring(color)`.
2159     ---
2160     ---@param ?string format One of: `#fff`, `#ffff`, `#ffffff`, `#ffffffff`,

```

```

2161 -- rgb, rgba, hsv, hsva, hsl, hsla, hwb, hwba, ncol, cmyk
2162 ---
2163 ---@return string
2164 ---
2165 ---@see Color:..tostring
2166 function Color:tostring(format)
2167     if format == nil then
2168         return tostring(self)
2169     end
2170
2171     format = format:lower()
2172
2173     if format:sub(1, 1) == '#' then
2174         if #format == 4 then
2175             return string.format('%#%x%x%x', utils.round(self.r * 0xf),
2176                 utils.round(self.g * 0xf), utils.round(self.b * 0xf))
2177         elseif #format == 5 then
2178             return string.format('%#%x%x%x%x', utils.round(self.r * 0xf),
2179                 utils.round(self.g * 0xf), utils.round(self.b * 0xf),
2180                 utils.round(self.a * 0xf))
2181         elseif #format == 7 then
2182             return string.format('%#%02x%02x%02x',
2183                 utils.round(self.r * 0xff), utils.round(self.g * 0xff),
2184                 utils.round(self.b * 0xff))
2185         elseif #format == 9 then
2186             return string.format('%#%02x%02x%02x%02x',
2187                 utils.round(self.r * 0xff), utils.round(self.g * 0xff),
2188                 utils.round(self.b * 0xff), utils.round(self.a * 0xff))
2189         end
2190     elseif format == 'rgb' then
2191         return string.format('rgb(%d, %d, %d)',
2192             utils.round(self.r * 0xff), utils.round(self.g * 0xff),
2193             utils.round(self.b * 0xff))
2194     elseif format == 'rgba' then
2195         return string.format('rgba(%d, %d, %d, %s)',
2196             utils.round(self.r * 0xff), utils.round(self.g * 0xff),
2197             utils.round(self.b * 0xff), self.a)
2198     elseif format == 'hsv' then
2199         local h, s, v = self:hsv()
2200         return string.format('hsv(%d, %d%%, %d%%)', utils.round(h * 360),
2201             utils.round(s * 100), utils.round(v * 100))
2202     elseif format == 'hsva' then
2203         local h, s, v, a = self:hsva()
2204         return string.format('hsva(%d, %d%%, %d%%, %s)',
2205             utils.round(h * 360), utils.round(s * 100),
2206             utils.round(v * 100), a)
2207     elseif format == 'hsl' then
2208         local h, s, l = self:hsl()
2209         return string.format('hsl(%d, %d%%, %d%%)', utils.round(h * 360),
2210             utils.round(s * 100), utils.round(l * 100))
2211     elseif format == 'hsla' then
2212         local h, s, l, a = self:hsla()
2213         return string.format('hsla(%d, %d%%, %d%%, %s)',
2214             utils.round(h * 360), utils.round(s * 100),
2215             utils.round(l * 100), a)
2216     elseif format == 'hwb' then
2217         local h, w, b = self:hwb()
2218         return string.format('hwb(%d, %d%%, %d%%)', utils.round(h * 360),
2219             utils.round(w * 100), utils.round(b * 100))
2220     elseif format == 'hwba' then
2221         local h, w, b, a = self:hwba()
2222         return string.format('hwba(%d, %d%%, %d%%, %s)',
2223             utils.round(h * 360), utils.round(w * 100),
2224             utils.round(b * 100), a)

```

```

2225     elseif format == 'ncol' then
2226         local h, w, b = self:hw()
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 ---

```

```

2289   ---@param Color other
2290   ---
2291   ---@return boolean self is darker than other
2292   function Color:lt(other)
2293     local _, _, la = self:hsl()
2294     local _, _, lb = other:hsl()
2295     return la < lb
2296   end
2297
2298   ---Checks whether color is as dark or darker.
2299   ---
2300   ---@param Color other
2301   ---
2302   ---@return boolean self is as dark or darker than other
2303   function Color:le(other)
2304     local _, _, la = self:hsl()
2305     local _, _, lb = other:hsl()
2306     return la <= lb
2307   end
2308
2309   ---Iterate through color.
2310   ---
2311   -- Iterates through r, g, b, and a.
2312   function Color:pairs()
2313     local function iter(tbl, k)
2314       if k == nil then
2315         return 'r', self.r
2316       elseif k == 'r' then
2317         return 'g', self.g
2318       elseif k == 'g' then
2319         return 'b', self.b
2320       elseif k == 'b' then
2321         return 'a', self.a
2322       end
2323     end
2324
2325     return iter, self, nil
2326   end
2327
2328   ---Get inverted clone of color.
2329   ---
2330   ---@return Color
2331   function Color:unm()
2332     return Color(self):invert()
2333   end
2334
2335   ---Mix two colors evenly.
2336   ---
2337   ---@param Color a first color
2338   ---@param Color b second color
2339   ---
2340   ---@return Color new color
2341   ---
2342   ---@see Color:mix
2343   function Color:__add(a, b)
2344     assert(Color.isColor(a) and Color.isColor(b),
2345            'Can only add two colors.')
2346     return Color(a):mix(b)
2347   end
2348
2349   ---Complement of even mix.
2350   ---
2351   ---@param Color a first color
2352   ---@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
2372     ↪ channel
2373     function Color.__band(a, b)
2374         local color, mask
2375         if Color.isColor(a) and type(b) == 'number' then
2376             color = a
2377             mask = b
2378         elseif Color.isColor(b) and type(a) == 'number' then
2379             color = b
2380             mask = a
2381         elseif Color.isColor(a) and Color.isColor(b) then
2382             color = a
2383             mask = bit_lshift(utils.round(b.r * 0xff), 16) +
2384                   bit_lshift(utils.round(b.g * 0xff), 8) +
2385                   utils.round(b.b * 0xff)
2386         else
2387             error(
2388                 'Required arguments: Color|number,Color|number Received: ' ..
2389                 type(a) .. ', ' .. type(b))
2390         end
2391
2392         return Color {
2393             bit_and(utils.round(color.r * 0xff), bit_rshift(mask, 16)) / 0xff,
2394             bit_and(utils.round(color.g * 0xff), bit_rshift(mask, 8)) / 0xff,
2395             bit_and(utils.round(color.b * 0xff), mask) / 0xff,
2396             color.a,
2397         }
2398     end
2399
2400     ---Apply rgb mask to color, providing backwards compatibility for Lua 5.1 and
2401     ↪ LuaJIT 2.1.0-beta3 (e.g. inside Neovim), which don't provide native support
2402     ↪ for bitwise operators.
2403     ---
2404     ---@param a Color/number # color or mask
2405     ---@param b Color/number # color or mask (if a and b are colors b is used as mask)
2406     ---
2407     ---@return Color new color
2408     ---
2409     ---@usage local new_col = Color.band(color, 0xff00ff) -- get new color without the
2410     ↪ green channel
2411     function Color.band(a, b)
2412         return Color.__band(a, b)
2413     end
2414
2415     ---Check whether `color` is a Color.
2416     ---

```



```

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
2446     ↪ PdfColorstackWhatsitNode]]
2447     if command == 'set' then
2448         whatsit.command = 0
2449     elseif command == 'push' then
2450         whatsit.command = 1
2451     elseif command == 'pop' then
2452         whatsit.command = 2
2453     elseif command == 'current' then
2454         whatsit.command = 3
2455     end
2456     if command ~= 'pop' then
2457         whatsit.data = self:format_pdf_colorstack_string()
2458     end
2459     return whatsit
2460 end
2461
2462 ---Write a PDF colorstock node using `node.write()`.
2463 ---
2464 ---@param command "set"/"push"/"pop"/"current"
2465 ---
2466 function Color:write_pdf_colorstack_node(command)
2467     node.write(self:create_pdf_colorstack_node(command))
2468 end
2469
2470 function Color:write_box()
2471     self:write_pdf_colorstack_node('push')
2472     local rule = node.new('rule') --[[@as RuleNode]]
2473     rule.width = tex.sp('0.5cm')
2474     rule.height = tex.sp('0.5cm')
2475     node.write(rule)
2476     self:write_pdf_colorstack_node('pop')

```

```

2476     end
2477
2478     return Color
2479
2480 end()
2481
2482 ---
2483 ---@param scheme 'base'|'svg'|'x11'
2484 local function print_color_table(scheme)
2485     for _, name in pairs(schemes[scheme]) do
2486         -- local color = Color({ r = rgb[1], g = rgb[2], b = rgb[3] })
2487         local color = colors[name]
2488         local r = utils.round(color[1] * 255)
2489         local g = utils.round(color[2] * 255)
2490         local b = utils.round(color[3] * 255)
2491         tex.print('\par\noindent')
2492         tex.print(string.format('\FarbeBox{rgba(%d, %d, %d, 1}', r, g, b))
2493         tex.print('\quad{}' .. name)
2494     end
2495 end
2496
2497 ---
2498 ---@param operator string # The PDF color operator, e. g. `0.2 0.5 1 rg 0.2 0.5 1
↪ RG`
2499 ---
2500 ---@return Color/nil
2501 local function convert_pdf_color_operator(operator)
2502     operator = operator:lower()
2503
2504     ---
2505     ---@param count_floats integer
2506     ---@param suffix string
2507     ---
2508     ---@return string
2509     local function build_pattern(count_floats, suffix)
2510         local pattern_float = '%d*%.?%d+'
2511
2512         local patterns = {}
2513         for i = 1, count_floats, 1 do
2514             patterns[i] = pattern_float
2515         end
2516         patterns[count_floats + 1] = suffix
2517
2518         return table.concat(patterns, ' +')
2519     end
2520
2521     local n = tonumber
2522
2523     local r, g, b = operator:match(build_pattern(3, 'rg'))
2524     if r ~= nil then
2525         log.debug('operator to RGB: %s %s %s', r, g, b)
2526         return Color({ r = n(r), g = n(g), b = n(b) })
2527     end
2528
2529     local c, m, y, k = operator:match(build_pattern(3, 'k'))
2530     if c ~= nil then
2531         log.debug('operator to CMYK: %s %s %s %s', c, m, y, k)
2532         return Color({ c = n(c), m = n(m), y = n(y), k = n(k) })
2533     end
2534
2535     local gray = operator:match(build_pattern(1, 'g'))
2536     if gray ~= nil then
2537         log.debug('operator to GRAY: %s', gray)
2538         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

```
1 %% farbe.tex
2 %% Copyright 2025 Josef Friedrich
3 %
4 % This work may be distributed and/or modified under the
5 % conditions of the LaTeX Project Public License, either version 1.3c
6 % of this license or (at your option) any later version.
7 % The latest version of this license is in
8 % http://www.latex-project.org/lppl.txt
9 % and version 1.3c or later is part of all distributions of LaTeX
10 % version 2008/05/04 or later.
11 %
12 % This work has the LPPL maintenance status `maintained'.
13 %
14 % The Current Maintainer of this work is Josef Friedrich.
15 %
16 % This work consists of the files farbe.lua, farbe.tex,
17 % and farbe.sty.
18
19 \directlua
20 {
21   if farbe == nil then
22     farbe = require('farbe')
23   end
24 }
25
26 \def\FarbePdfLiteral#1{\csname\string\color@#1\endcsname}
27
28 \def\FarbeImport#1{%
29   \directlua
30   {
31     farbe.import_color('#1', '\FarbePdfLiteral{#1}')
32   }%
33 }
34
35 \def\FarbeColor#1
36 {%
37   \directlua{farbe.Color('#1'):write_pdf_colorstack_node('push')}}%
38 }
39
40 \def\FarbeTextColor#1#2
41 {%
42   \protect\leavevmode{%
43     \directlua{farbe.Color('#1'):write_pdf_colorstack_node('push')}}%
44   #2%
45   \directlua{farbe.Color('#1'):write_pdf_colorstack_node('pop')}}%
46 }
47 }
48
49 \def\FarbeBox#1
50 {%
51   \directlua{farbe.Color('#1'):write_box()}}%
52 }
```

4.3 farbe.sty

```
1 %% farbe.sty
2 %% Copyright 2025 Josef Friedrich
3 %
4 % This work may be distributed and/or modified under the
5 % conditions of the LaTeX Project Public License, either version 1.3c
6 % of this license or (at your option) any later version.
7 % The latest version of this license is in
8 % http://www.latex-project.org/lppl.txt
9 % and version 1.3c or later is part of all distributions of LaTeX
10 % version 2008/05/04 or later.
11 %
12 % This work has the LPPL maintenance status `maintained'.
13 %
14 % The Current Maintainer of this work is Josef Friedrich.
15 %
16 % This work consists of the files farbe.lua, farbe.tex,
17 % and farbe.sty.
18
19 \NeedsTeXFormat{LaTeX2e}
20 \ProvidesPackage{farbe}[2025/05/31 v0.1.0 Color management (conversion, names) for
   ⇨ LuaTeX implemented in Lua.]
21
22 \input farbe.tex
```