Package 'shades'

July 2, 2026

Version 1.5.0

Date 2026-07-01

Title Simple Colour Manipulation

Description
Functions for easily manipulating colours, creating colour scales and calculating colour distances.

Suggests tinytest, covr

Enhances colorspace, ggplot2, tinyplot

Encoding UTF-8

License BSD_3_clause + file LICENCE

URL https://github.com/jonclayden/shades

BugReports https://github.com/jonclayden/shades/issues

Config/roxygen2/version 8.0.0

Config/roxygen2/markdown TRUE

NeedsCompilation no

Author Jon Clayden [cre, aut] (ORCID: <https://orcid.org/0000-0002-6608-0619>)

Maintainer Jon Clayden <code@clayden.org>

Repository CRAN

Date/Publication 2026-07-02 21:50:07 UTC

Contents

addmix 2
complement 3
coords 3
dichromat 4
distance 5
expect_equal_shades 6
gradient 6
saturation 7
shade 9

space 11
swatch 12
warp 13

Index **15**

`addmix` *Colour mixtures*

Description

These functions allow colours to be mixed in any colour space, either additively (like light) or subtractively (like paint). The infix form `%.)%` is an alternative for `addmix`, and `%_/%` for `submix`, with the mixing amount being fixed to 1 in these cases.

Usage

```
addmix(base, mixer, amount = 1, space = NULL)

submix(base, mixer, amount = 1, space = NULL)

X %.)% Y

X %_/% Y
```

Arguments

`base`, `X`	A vector of base colours, or a palette function or scale.
`mixer`, `Y`	A vector of colours to mix in.
`amount`	The amount of each colour to mix in, relative to the amount of the base. This will be recycled to the length of `mixer`.
`space`	A string giving the space in which to perform the mixing, or `NULL`. In the latter case, the space of `base` will be used.

Value

New colours of class `"shade"`, or a new palette function.

Author(s)

Jon Clayden code@clayden.org

Examples

```
addmix(c("red","green","blue"), "red")
submix(c("cyan","magenta","yellow"), "cyan")
```

complement *Complementary colours*

Description

This function returns the complement of its argument, the "opposite" colours in the specified space.

Usage

```
complement(shades, space = NULL)
```

Arguments

shades	One or more colours, in any suitable form (see shade), or a palette function or scale.
space	The space in which to take the complement.

Value

New colours of class "shade", or a new palette function.

Examples

```
complement("cyan")
complement("cyan", space="HSV")
complement("cyan", space="Lab")
```

coords *Retrieve the coordinates of a colour vector*

Description

This function retrieves the coordinates of a colour vector's elements, within whatever space it is currently defined.

Usage

```
coords(x, ...)
```

Arguments

x	An R object.
...	Additional arguments to methods.

Value

A matrix giving colour coordinates in the relevant space, one colour per row. Columns are typically named.

Author(s)

Jon Clayden code@clayden.org

Examples

```
coords("red")
```

`dichromat` *Simulate colour appearance for dichromats*

Description

This functions manipulates colours to simulate the effects of different kinds of colour blindness, and specifically dichromacy, in which only two of the usual three types of photoreceptors are present. There are three types, corresponding to the loss of red, green or blue photoreceptors.

Usage

```
dichromat(shades, type = c("protanopic", "deuteranopic", "tritanopic", "none"))
```

Arguments

`shades`	One or more colours, in any suitable form (see `shade`).
`type`	The type of colour vision deficiency to simulate: protanopia (red blindness), deuteranopia (green blindness) or tritanopia (blue blindness). The latter is the rarest in the population. `"none"` is also available, as a pass-through option. Abbreviations, such as the first letter, may be used, and multiple values are acceptable.

Value

New colours of class `"shade"` in LMS space, representing projections of the original shades onto a submanifold appropriate to the type of dichromacy being simulated.

Author(s)

Jon Clayden code@clayden.org

References

Brettel, H., Viénot, F. and Mollon, J.D. (1997). Computerized simulation of color appearance for dichromats. Journal of the Optical Society of America A 14(10):2647-2655.

Examples

```
dichromat(c("red", "green", "blue"))
```

`distance` *Colour distance and contrast*

Description

The `distance()` function calculates a distance measure that aims to quantify perceptual difference between a vector of colours and a reference colour. The measure in question is the CIE Delta E (2000), which is calculated based on colour coordinates in Lab space. Colour contrast relates more specifically to legibility, in the context where text in one colour is overlaid on a background in another. The W3C definition used here ranges between 1 (identical colours, no contrast) and 21 (black on white, maximum contrast).

Usage

```
distance(shades, reference)

contrast(shades, reference)
```

Arguments

`shades`	One or more colours, in any suitable form (see `shade()`).
`reference`	A single reference colour.

Value

A numeric vector of distances or contrasts.

Author(s)

Jon Clayden code@clayden.org

References

See `http://www.brucelindbloom.com/index.html?Eqn_DeltaE_CIE2000.html` for CIE Delta E. The contrast measure is defined as part of the W3C Web Content Accessibility Guidelines, version 2.0, detailed at `https://www.w3.org/TR/WCAG20/`.

Examples

```
distance(c("red","green","blue"), "red")
contrast(c("red","green","blue"), "red")
```

expect_equal_shades *Expectation for comparing shades*

Description

This function provides an expectation for use with the `tinytest` package, which compares two colour vectors via `all.equal.shade(hexonly=TRUE)`. It evaluates `TRUE` if the vectors have the same length and resolve to the same hex colour strings.

Usage

```
expect_equal_shades(current, target, ..., info = NA_character_)
```

Arguments

`current`	The colour vector to test.
`target`	The target values to compare against, in any form valid for casting to `shade()`.
`...`	Further arguments to `all.equal()`.
`info`	An optional information string shown in case of failure.

Value

A `"tinytest"` object.

gradient *Colour gradients*

Description

This function returns a set of colours interpolating between the specified key colours, equally separated in the specified space.

Usage

```
gradient(shades, steps = NULL, space = NULL)
```

Arguments

`shades`	Two or more colours, in any suitable form (see `shade`), or a named colour map such as `"viridis"`.
`steps`	An integer giving the number of shades required in the palette. If `NULL`, a function will instead be returned that takes this argument.
`space`	The colour space to traverse. Defaults to the current space of `shades`, or `"Lab"` for the `matplotlib` colour maps, or `"sRGB"` otherwise.

Details

The key colours may be specified explicitly, or else a built-in colour map may be used. The maps available are currently those developed for Python's `matplotlib` 2.0, namely `"magma"`, `"inferno"`, `"plasma"` and `"viridis"`, and certain ColorBrewer palettes, namely `"Blues"`, `"Reds"`, `"YlOrRd"` (yellow-orange-red) and `"RdBu"` (red-grey-blue, a balanced diverging scale).

Value

A character vector of class `"shade"` containing the gradient elements in the specified space, or a palette function.

Author(s)

Jon Clayden code@clayden.org

References

https://bids.github.io/colormap/ for the `matplotlib` colour maps; https://colorbrewer2.org for the ColorBrewer ones.

Examples

```
gradient(c("red","blue"), 5)
gradient(c("red","blue"), 5, space="Lab")
gradient("viridis", 5)
```

`saturation` *Query or change colour properties*

Description

These functions obtain the value of a colour property, or modify it. They will convert between colour spaces as required, but the RGB representation will be appropriately updated in the result.

Usage

```
saturation(shades, values = NULL)

brightness(shades, values = NULL)

lightness(shades, values = NULL)

luminance(shades, values = NULL)

chroma(shades, values = NULL)

hue(shades, values = NULL)
```

```
opacity(shades, values = NULL)

delta(...)

scalefac(...)

recycle(...)
```

Arguments

shades	One or more colours, in any suitable form (see shade), or a palette function or scale.
values	New values for the property in question, with NA as a pass-through value that will leave the property as-is. If NULL, the current value(s) will be returned. May also be a function computing new values from old ones, such as delta, which adds its argument, or scalefac, which multiplies it.
...	Arguments to replacement functions delta, scalefac and recycle, which will be concatenated.

Details

Brightness and lightness differ technically, in the sense that one is absolute and the other is relative. Intuitively, a grey square on white paper is brighter under bright sunlight than in a dark room, but its lightness (relative to the white around it) is constant between conditions. In these functions, brightness is "value" in HSV space and is between 0 and 1, while lightness is defined in Lab space and is between 0 and 100. Luminance is different again, being more physically based and defined in XYZ space. Saturation and chroma are also related. Hue is defined in HSV space, with red at 0º (and 360º), which is the most familiar parameterisation.

Value

Current colour property values, or new colours of class "shade". If shades is a function, the result will be a new function that wraps the old one and modifies its return value accordingly.

Note

The colour property functions are vectorised over both of their arguments, such that the dimensions of the result will be c(length(values),dim(shades)). However, the recycle function can be used to suppress the usual dimensional expansion, and instead follow R's standard recycling rule.

Author(s)

Jon Clayden code@clayden.org

Examples

```
saturation(c("papayawhip","lavenderblush","olivedrab"))
```

```
saturation("papayawhip", 0.7)
saturation("papayawhip", delta(0.2))
saturation("papayawhip", scalefac(1.5))

saturation(c("red","green"), c(0.4,0.6))
saturation(c("red","green"), recycle(0.4,0.6))
```

---

shade *The shade class*

Description

Objects of class "shade" are simply standard R character vectors representing one or more 8-bit (s)RGB colours in CSS-like hex format, but with extra attributes giving the current colour space and coordinates. Opacity values are also supported.

Usage

```
shade(x, ...)

## S3 method for class 'shade'
shade(x, ...)

## S3 method for class 'color'
shade(x, ...)

## S3 method for class 'matrix'
shade(x, space = "sRGB", alpha = NULL, ...)

## S3 method for class 'character'
shade(x, ...)

## Default S3 method:
shade(x, ...)

shades(...)

## S3 method for class 'shade'
print(x, ...)

## S3 method for class 'shade'
x[i]

## S3 replacement method for class 'shade'
x[i] <- value

## S3 method for class 'shade'
c(...)
```

```
## S3 method for class 'shade'
rep(x, ...)

## S3 method for class 'shade'
rev(x)

## S3 method for class 'shade'
x == y

## S3 method for class 'shade'
x != y

## S3 method for class 'shade'
all.equal(target, current, hexonly = FALSE, ...)
```

Arguments

x, y	R objects, or "shade" objects for methods.
...	For shade(), additional parameters to methods. For shades() and c(), any number of colours in any acceptable form.
space	For a matrix, the space in which coordinates are being provided.
alpha	For a matrix, an associated vector of opacity values between 0 and 1, if required.
i	An index vector.
value	A vector of replacement colours.
target, current	Shade vectors to compare.
hexonly	If TRUE, compare only on the basis of the hex strings. Otherwise test for equal coordinates.

Details

The shade() function creates a colour vector from a single main argument plus method-specific arguments where appropriate. The shades() function provides an alternative interface that converts each of its arguments to shades, with direct inline naming, and then concatenates them.

Comparison between "shade" objects x and y is achieved by converting y (the second argument) into the colour space of x and then comparing coordinates, after any clipping.

Value

A character vector of class "shade", with additional attributes as follows.

space	A string naming a colour space.
coords	A matrix giving colour coordinates in the relevant space, one colour per row.
alpha	A numeric vector of alpha (opacity) values between 0 and 1, one value per element of the vector. If not present then full opacity is assumed for every element.
names	A character vector of element names, in the usual R convention. If not present then the object is unnamed.

Note

When concatenating, shades that are all from the same space will remain in that space, but shades from different spaces will be warped to "XYZ" space.

Author(s)

Jon Clayden code@clayden.org

Examples

```
s <- shade(c("red", "green", "blue"))
s[1]
s[1] <- "pink"
```

---

space *Retrieve the space of a colour vector*

Description

This function retrieves the colour space in which its argument is currently defined.

Usage

```
space(x, ...)
```

Arguments

x	An R object.
...	Additional arguments to methods.

Value

A string naming a colour space.

Author(s)

Jon Clayden code@clayden.org

Examples

```
space("red")
```

swatch *Simple colour swatches*

Description

This function provides a simple visualisation of a colour series as a series of boxes against the specified background colour. If the input has more than one dimension then the boxes will be arranged in a grid (flattening further dimensions after the second).

Usage

```
swatch(x, bg = "white", border = "grey50", cex = 0.7, font = 2, ...)
```

Arguments

x	One or more colours, in any suitable form (see shade()).
bg	A background colour.
border	The border colour to draw around each box.
cex	The font size to use for labelling named shades.
font	The font weight to use for labelling named shades, in the sense of par(). The default, 2, corresponds to boldface.
...	Additional arguments (currently unused).

Author(s)

Jon Clayden code@clayden.org

See Also

par()

Examples

```
swatch(c("red", "green", "blue"))
```

warp *Shift colours between spaces*

Description

This function shifts the current colour space of its arguments to the specified space, returning a new object of class "shade".

Usage

```
warp(x, space)
```

Arguments

x	An R object which can be coerced to class "shade".
space	A string naming the new space.

Details

Valid names for spaces are currently those supported by the convertColor function, namely "sRGB", "Apple RGB", "CIE RGB", "XYZ", "Lab" and "Luv"; plus "RGB" (which is treated as an alias for "sRGB"), "HSV", "LCh", "LMS" and "Oklab". Case is not significant.

Value

A new object of class "shade".

Note

LMS space, used for chromatic adaptation and simulating colour blindness, is not uniquely defined. Here we use the (linearised) Bradford transform, obtained by Lam (1985) and used widely in ICC colour profiles and elsewhere, to transform to and from CIE XYZ space.

R uses the D65 standard illuminant as the reference white for the "Lab" and "Luv" spaces.

Author(s)

Jon Clayden code@clayden.org

References

The Oklab colour space is explained at https://bottosson.github.io/posts/oklab/.

Lam, K.M. (1985). Metamerism and colour constancy. PhD thesis, University of Bradford.

See Also

convertColor

Examples

```
warp("red", "HSV")
```

Index

!=.shade *(shade)*, 9
==.shade *(shade)*, 9
[.shade *(shade)*, 9
[<-.shade *(shade)*, 9
%.)% *(addmix)*, 2
%_/% *(addmix)*, 2

addmix, 2
all.equal(), *6*
all.equal.shade *(shade)*, 9

brightness *(saturation)*, 7

c.shade *(shade)*, 9
chroma *(saturation)*, 7
complement, 3
contrast *(distance)*, 5
convertColor, *13*
coords, 3

delta *(saturation)*, 7
dichromat, 4
distance, 5

expect_equal_shades, 6

gradient, 6

hue *(saturation)*, 7

lightness *(saturation)*, 7
luminance *(saturation)*, 7

opacity *(saturation)*, 7

par(), *12*
print.shade *(shade)*, 9

recycle *(saturation)*, 7
rep.shade *(shade)*, 9
rev.shade *(shade)*, 9

saturation, 7
scalefac *(saturation)*, 7
shade, *3*, *4*, *6*, *8*, 9
shade(), *5*, *6*, *12*
shades *(shade)*, 9
space, 11
submix *(addmix)*, 2
swatch, 12

warp, 13