Package ‘scales’

August 9, 2018

Title Scale Functions for Visualization
Version 1.0.0
Description Graphical scales map data to aesthetics, and provide methods for automatically determining breaks and labels for axes and legends.
License MIT + file LICENSE
BugReports https://github.com/r-lib/scales/issues
Depends R (>= 3.1)
Imports labeling, munsell (>= 0.5), R6, RColorBrewer, Rcpp, viridisLite
Suggests dichromat, bit64, covr, hms, testthat (>= 2.0)
LinkingTo Rcpp
LazyLoad yes
RoxygenNote 6.0.1.9000
Encoding UTF-8
NeedsCompilation yes
Author Hadley Wickham [aut, cre], RStudio [cph]
Maintainer Hadley Wickham <hadley@rstudio.com>
Repository CRAN
Date/Publication 2018-08-09 10:10:03 UTC

R topics documented:

- abs_area ................................................................. 3
- alpha ......................................................................... 4
- area_pal ..................................................................... 4
- as.trans .................................................................... 5
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>asn_trans</td>
<td>5</td>
</tr>
<tr>
<td>atanh_trans</td>
<td>5</td>
</tr>
<tr>
<td>boxcox_trans</td>
<td>6</td>
</tr>
<tr>
<td>brewer_pal</td>
<td>7</td>
</tr>
<tr>
<td>cbreaks</td>
<td>7</td>
</tr>
<tr>
<td>censor</td>
<td>8</td>
</tr>
<tr>
<td>col2hcl</td>
<td>9</td>
</tr>
<tr>
<td>colour_ramp</td>
<td>9</td>
</tr>
<tr>
<td>col_numeric</td>
<td>10</td>
</tr>
<tr>
<td>cscale</td>
<td>12</td>
</tr>
<tr>
<td>date_breaks</td>
<td>13</td>
</tr>
<tr>
<td>date_format</td>
<td>13</td>
</tr>
<tr>
<td>date_trans</td>
<td>14</td>
</tr>
<tr>
<td>dichromat_pal</td>
<td>14</td>
</tr>
<tr>
<td>discard</td>
<td>15</td>
</tr>
<tr>
<td>div_gradient_pal</td>
<td>15</td>
</tr>
<tr>
<td>dollar_format</td>
<td>16</td>
</tr>
<tr>
<td>dscale</td>
<td>17</td>
</tr>
<tr>
<td>expand_range</td>
<td>18</td>
</tr>
<tr>
<td>exp_trans</td>
<td>18</td>
</tr>
<tr>
<td>extended_breaks</td>
<td>19</td>
</tr>
<tr>
<td>format_format</td>
<td>19</td>
</tr>
<tr>
<td>gradient_n_pal</td>
<td>20</td>
</tr>
<tr>
<td>grey_pal</td>
<td>20</td>
</tr>
<tr>
<td>hms_trans</td>
<td>21</td>
</tr>
<tr>
<td>hue_pal</td>
<td>21</td>
</tr>
<tr>
<td>identity_pal</td>
<td>22</td>
</tr>
<tr>
<td>identity_trans</td>
<td>22</td>
</tr>
<tr>
<td>linetype_pal</td>
<td>22</td>
</tr>
<tr>
<td>log1p_trans</td>
<td>23</td>
</tr>
<tr>
<td>log_breaks</td>
<td>23</td>
</tr>
<tr>
<td>log_trans</td>
<td>24</td>
</tr>
<tr>
<td>manual_pal</td>
<td>24</td>
</tr>
<tr>
<td>math_format</td>
<td>24</td>
</tr>
<tr>
<td>muted</td>
<td>25</td>
</tr>
<tr>
<td>number_bytes_format</td>
<td>26</td>
</tr>
<tr>
<td>number_format</td>
<td>27</td>
</tr>
<tr>
<td>ordinal_format</td>
<td>29</td>
</tr>
<tr>
<td>package-scales</td>
<td>30</td>
</tr>
<tr>
<td>parse_format</td>
<td>30</td>
</tr>
<tr>
<td>pretty_breaks</td>
<td>31</td>
</tr>
<tr>
<td>probability_trans</td>
<td>31</td>
</tr>
<tr>
<td>pseudo_log_trans</td>
<td>32</td>
</tr>
<tr>
<td>pvalue_format</td>
<td>32</td>
</tr>
<tr>
<td>Range</td>
<td>33</td>
</tr>
<tr>
<td>reciprocal_trans</td>
<td>33</td>
</tr>
<tr>
<td>regular_minor_breaks</td>
<td>34</td>
</tr>
<tr>
<td>rescale</td>
<td>34</td>
</tr>
</tbody>
</table>
**abs_area**

Point area palette (continuous), with area proportional to value.

**Description**

Point area palette (continuous), with area proportional to value.

**Usage**

```r
abs_area(max)
```

**Arguments**

- `max` A number representing the maximum size.
alpha

Modify colour transparency. Vectorised in both colour and alpha.

Description

Modify colour transparency. Vectorised in both colour and alpha.

Usage

alpha(colour, alpha = NA)

Arguments

colour
colour

alpha
new alpha level in [0,1]. If alpha is NA, existing alpha values are preserved.

Examples

alpha("red", 0.1)
alpha(colours(), 0.5)
alpha("red", seq(0, 1, length.out = 10))

area_pal

Point area palette (continuous).

Description

Point area palette (continuous).

Usage

area_pal(range = c(1, 6))

Arguments

range

Numeric vector of length two, giving range of possible sizes. Should be greater than 0.
### `as.trans`

**Description**
Convert character string to transformer.

**Usage**
```r
as.trans(x)
```

**Arguments**
- `x` name of transformer

### `asn_trans`

**Description**
Arc-sin square root transformation.

**Usage**
```r
asn_trans()
```

### `atanh_trans`

**Description**
Arc-tangent transformation.

**Usage**
```r
atanh_trans()
```
Description

The Box-Cox transformation is a flexible transformation, often used to transform data towards normality. The modulus transformation generalises Box-Cox to also work with negative values.

Usage

```r
boxcox_trans(p, offset = 0)
modulus_trans(p, offset = 1)
```

Arguments

- `p` Transformation exponent, \( \lambda \).
- `offset` Constant offset. \( 0 \) for Box-Cox type 1, otherwise any non-negative constant (Box-Cox type 2). `modulus_trans()` sets the default to 1.

Details

The Box-Cox power transformation (type 1) requires strictly positive values and takes the following form for \( y > 0 \):

\[
y^{(\lambda)} = \frac{y^\lambda - 1}{\lambda}
\]

When \( y = 0 \), the natural log transform is used.

The modulus transformation implements a generalisation of the Box-Cox transformation that works for data with both positive and negative values. The equation takes the following forms, when \( y \neq 0 \):

\[
y^{(\lambda)} = \text{sign}(y) \times \frac{(|y| + 1)^\lambda - 1}{\lambda}
\]

and when \( y = 0 \):

\[
y^{(\lambda)} = \text{sign}(y) \times \ln(|y| + 1)
\]

References


brewer_pal  

Colour Brewer palette (discrete).

Description

Colour Brewer palette (discrete).

Usage

```r
brewer_pal(type = "seq", palette = 1, direction = 1)
```

Arguments

- `type` One of seq (sequential), div (diverging) or qual (qualitative)
- `palette` If a string, will use that named palette. If a number, will index into the list of palettes of appropriate type
- `direction` Sets the order of colours in the scale. If 1, the default, colours are as output by `RColorBrewer::brewer_pal()`. If -1, the order of colours is reversed.

References

http://colorbrewer2.org

Examples

```r
show_col(brewer_pal()(10))
show_col(brewer_pal("div")(5))
show_col(brewer_pal(palette = "Greens")(5))

# Can use with gradient_n to create a continous gradient
cols <- brewer_pal("div")(5)
show_col(gradient_n_pal(cols)(seq(0, 1, length.out = 30)))
```

cbreaks  

Compute breaks for continuous scale.

Description

This function wraps up the components needed to go from a continuous range to a set of breaks and labels suitable for display on axes or legends.

Usage

```r
cbreaks(range, breaks = extended_breaks(),
       labels = scientific_format())
```
Arguments

- **range**: numeric vector of length 2 giving the range of the underlying data
- **breaks**: either a vector of break values, or a break function that will make a vector of breaks when given the range of the data
- **labels**: either a vector of labels (character vector or list of expression) or a format function that will make a vector of labels when called with a vector of breaks. Labels can only be specified manually if breaks are - it is extremely dangerous to supply labels if you don’t know what the breaks will be.

Examples

```r
cbreaks(c(0, 100))
cbreaks(c(0, 100), pretty_breaks(3))
cbreaks(c(0, 100), pretty_breaks(10))
cbreaks(c(1, 100), log_breaks())
cbreaks(c(1, 1e4), log_breaks())

cbreaks(c(0, 100), labels = math_format())
cbreaks(c(0, 1), labels = percent_format())
cbreaks(c(0, 1e6), labels = comma_format())
cbreaks(c(0, 1e6), labels = dollar_format())

# You can also specify them manually:
cbreaks(c(0, 100), breaks = c(15, 20, 80))
cbreaks(c(0, 100), breaks = c(15, 20, 80), labels = c(1.5, 2.0, 8.0))
cbreaks(c(0, 100), breaks = c(15, 20, 80),
       labels = expression(alpha, beta, gamma))
```

censor

* Censor any values outside of range.

Description

Censor any values outside of range.

Usage

```r
censor(x, range = c(0, 1), only.finite = TRUE)
```

Arguments

- **x**: numeric vector of values to manipulate.
- **range**: numeric vector of length two giving desired output range.
- **only.finite**: if TRUE (the default), will only modify finite values.

Examples

```r
censor(c(-1, 0.5, 1, 2, NA))
```
col2hcl

Modify standard R colour in hcl colour space.

Description
Transforms rgb to hcl, sets non-missing arguments and then backtransforms to rgb.

Usage
col2hcl(colour, h, c, l, alpha = 1)

Arguments
- colour: character vector of colours to be modified
- h: new hue
- c: new chroma
- l: new luminance
- alpha: alpha value. Defaults to 1.

Examples
col2hcl(colors())

colour_ramp
Fast colour interpolation

Description
Returns a function that maps the interval [0,1] to a set of colours. Interpolation is performed in the CIELAB colour space. Similar to colorRamp(space = 'Lab'), but hundreds of times faster, and provides results in "#RRGGBB" (or "#RRGGBBA" character form instead of RGB colour matrices.

Usage
colour_ramp(colors, na.color = NA, alpha = TRUE)

Arguments
- colors: Colours to interpolate; must be a valid argument to grDevices::col2rgb(). This can be a character vector of "#RRGGBB" or "#RRGGBBA", colour names from grDevices::colors(), or a positive integer that indexes into grDevices::palette().
- na.color: The colour to map to NA values (for example, "#606060" for dark grey, or "#00000000" for transparent) and values outside of [0,1]. Can itself by NA, which will simply cause an NA to be inserted into the output.
Whether to include alpha transparency channels in interpolation. If TRUE then the alpha information is included in the interpolation. The returned colours will be provided in "#RRGGBBAA" format when needed, i.e., in cases where the colour is not fully opaque, so that the "AA" part is not equal to "FF". Fully opaque colours will be returned in "#RRGGBB" format. If FALSE, the alpha information is discarded before interpolation and colours are always returned as "#RRGGBB".

**Value**

A function that takes a numeric vector and returns a character vector of the same length with RGB or RGBA hex colours.

**See Also**

colorRamp

---

### col_numeric

**Colour mapping**

**Description**

Conveniently maps data values (numeric or factor/character) to colours according to a given palette, which can be provided in a variety of formats.

**Usage**

```r
col_numeric(palette, domain, na.color = "#808080")
col_bin(palette, domain, bins = 7, pretty = TRUE,
    na.color = "#808080")
col_quantile(palette, domain, n = 4, probs = seq(0, 1, length.out = n + 1),
    na.color = "#808080")
col_factor(palette, domain, levels = NULL, ordered = FALSE,
    na.color = "#808080")
```

**Arguments**

- **palette**: The colours or colour function that values will be mapped to.
- **domain**: The possible values that can be mapped.

For `col_numeric` and `col_bin`, this can be a simple numeric range (e.g. `c(0, 100)`); `col_quantile` needs representative numeric data; and `col_factor` needs categorical data.

If `NULL`, then whenever the resulting colour function is called, the x value will represent the domain. This implies that if the function is invoked multiple times, the encoding between values and colours may not be consistent; if consistency is needed, you must provide a non-NULL domain.
**col_numeric**

na.color. The colour to return for NA values. Note that na.color = NA is valid.

bins. Either a numeric vector of two or more unique cut points or a single number (greater than or equal to 2) giving the number of intervals into which the domain values are to be cut.

pretty. Whether to use the function pretty() to generate the bins when the argument bins is a single number. When pretty = TRUE, the actual number of bins may not be the number of bins you specified. When pretty = FALSE, seq() is used to generate the bins and the breaks may not be "pretty".

n. Number of equal-size quantiles desired. For more precise control, use the probs argument instead.

probs. See stats::quantile(). If provided, the n argument is ignored.

levels. An alternate way of specifying levels; if specified, domain is ignored.

ordered. If TRUE and domain needs to be coerced to a factor, treat it as already in the correct order.

**Details**

col_numeric is a simple linear mapping from continuous numeric data to an interpolated palette.

col_bin also maps continuous numeric data, but performs binning based on value (see the base::cut() function).

col_quantile similarly bins numeric data, but via the stats::quantile() function.

col_factor maps factors to colours. If the palette is discrete and has a different number of colours than the number of factors, interpolation is used.

The palette argument can be any of the following:

1. A character vector of RGB or named colours. Examples: palette(c("#000000", "#0000FF", "#FFFFFF"),
   topo.colors(10)
2. The name of an RColorBrewer palette, e.g. "BuPu" or "Greens".
3. A function that receives a single value between 0 and 1 and returns a colour. Examples: colorRamp(c("#000000", "#FFFFFF"), interpolate="spline").

**Value**

A function that takes a single parameter x; when called with a vector of numbers (except for col_factor, which expects factors/characters), #RRGGBB colour strings are returned.

**Examples**

```r
col <- col_bin("Greens", domain = 0:100)
show_col(col(sort(runif(10, 60, 100))))

# Exponential distribution, mapped continuously
show_col(col_numeric("Blues", domain = NULL)(sort(rexp(16))))
# Exponential distribution, mapped by interval
show_col(col_bin("Blues", domain = NULL, bins = 4)(sort(rexp(16))))
# Exponential distribution, mapped by quantile
```
show_col(col_quantile("Blues", domain = NULL)(sort(rexp(16))))

# Categorical data; by default, the values being coloured span the gamut...
show_col(col_factor("RdYlBu", domain = NULL)(LETTERS[1:5]))
# ...unless the data is a factor, without droplevels...
show_col(col_factor("RdYlBu", domain = NULL)(factor(LETTERS[1:5], levels=LETTERS)))
# ...or the domain is stated explicitly.
show_col(col_factor("RdYlBu", levels = LETTERS)(LETTERS[1:5]))

---

cscale

**Continuous scale.**

**Description**

Continuous scale.

**Usage**

cscale(x, palette, na.value = NA_real_, trans = identity_trans())

**Arguments**

- **x**: vector of continuous values to scale
- **palette**: palette to use.
  Built in palettes: area_pal, brewer_pal, dichromat_pal, div_gradient_pal, gradient_n_pal, grey_pal, hue_pal, identity_pal, linetype_pal, manual_pal, rescale_pal, seq_gradient_pal, shape_pal, viridis_pal
- **na.value**: value to use for missing values
- **trans**: transformation object describing the how to transform the raw data prior to scaling. Defaults to the identity transformation which leaves the data unchanged.

**Examples**

with(mtcars, plot(disp, mpg, cex = cscale(hp, rescale_pal())))
with(mtcars, plot(disp, mpg, cex = cscale(hp, rescale_pal(),
  trans = sqrt_trans())))
with(mtcars, plot(disp, mpg, cex = cscale(hp, area_pal())))
with(mtcars, plot(disp, mpg, pch = 20, cex = 5,
  col = cscale(hp, seq_gradient_pal("grey80", "black"))))
date_breaks

Regularly spaced dates.

Description
Regularly spaced dates.

Usage
date_breaks(width = "1 month")

Arguments
width an interval specification, one of "sec", "min", "hour", "day", "week", "month", "year". Can be by an integer and a space, or followed by "s". Fractional seconds are supported.

date_format

Formatted dates and times.

Description
Formatted dates and times.

Usage
date_format(format = "%Y-%m-%d", tz = "UTC")
time_format(format = "%H:%M:%S", tz = "UTC")

Arguments
format Date/time format using standard POSIX specification. See strftime() for possible formats.
tz a time zone name, see timezones(). Defaults to UTC

Examples

a_time <- ISOdatetime(2012, 1, 11, 30, 0, tz = "UTC")
a_date <- as.Date(a_time)
date_format()(a_date)
date_format(format = "%A")(a_date)
time_format()(a_time)
time_format(tz = "Europe/Berlin")(a_time)


```
a_hms <- hms::as.hms(a_time, tz = "UTC")
time_format(format = "%H:%M")(a_hms)
```

---

**date_trans**  
*Transformation for dates (class Date).*

**Description**
Transformation for dates (class Date).

**Usage**
```
date_trans()
```

**Examples**
```
years <- seq(as.Date("1910/1/1"), as.Date("1999/1/1"), "years")
t <- date_trans()
t$transform(years)
t$inverse(t$transform(years))
t$format(t$breaks(range(years)))
```

---

**dichromat_pal**  
*Dichromat (colour-blind) palette (discrete).*

**Description**
Dichromat (colour-blind) palette (discrete).

**Usage**
```
dichromat_pal(name)
```

**Arguments**

- **name**  
  Name of colour palette. One of: *BrowntoBlue.10, BrowntoBlue.12, BluetoDarkOrange.12, BluetoDarkOrange.18, DarkRedtoBlue.12, DarkRedtoBlue.18, BluetoGreen.14, BluetoGray.8, BluetoOrangeRed.14, BluetoOrange.10, BluetoOrange.12, BluetoOrange.8, LightBluetoDarkBlue.10, LightBluetoDarkBlue.7, Categorical.12, GreentoMagenta.16, SteppedSequential.5*
Examples

```r
show_col(dichromat_pal("BluetoOrange.10")(10))
show_col(dichromat_pal("BluetoOrange.10")(5))

# Can use with gradient_n to create a continuous gradient
cols <- dichromat_pal("DarkRedtoBlue.12")(12)
show_col(gradient_n_pal(cols)(seq(0, 1, length.out = 30)))
```

---

**discard**

*Discard any values outside of range.*

**Description**

Discard any values outside of range.

**Usage**

```r
discard(x, range = c(0, 1))
```

**Arguments**

- **x** numeric vector of values to manipulate.
- **range** numeric vector of length two giving desired output range.

**Examples**

```r
discard(c(-1, 0.5, 1, 2, NA))
```

---

**div_gradient_pal**

*Diverging colour gradient (continuous).*

**Description**

Diverging colour gradient (continuous).

**Usage**

```r
div_gradient_pal(low = mns1("10B 4/6"), mid = mns1("N 8/0"),
                high = mns1("10R 4/6"), space = "Lab")
```

**Arguments**

- **low** colour for low end of gradient.
- **mid** colour for mid point
- **high** colour for high end of gradient.
- **space** colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.
Examples

```r
x <- seq(-1, 1, length.out = 100)
r <- sqrt(outer(x^2, x^2, "+"))
image(r, col = div_gradient_pal(seq(0, 1, length.out = 12)))
image(r, col = div_gradient_pal(seq(0, 1, length.out = 30)))
image(r, col = div_gradient_pal(seq(0, 1, length.out = 100)))
library(munsell)
image(r, col = div_gradient_pal(low = mnsl(complement("10R 4/6"), fix = TRUE)(seq(0, 1, length = 100)))
```

dollar_format

| Currency formatter: round to nearest cent and display dollar sign. |

Description

The returned function will format a vector of values as currency. If accuracy is not specified, values are rounded to the nearest cent, and cents are displayed if any of the values has a non-zero cents and the largest value is less than `largest_with_cents` which by default is 100,000.

Usage

```r
dollar_format(accuracy = NULL, scale = 1, prefix = "$",
             suffix = "", big.mark = ",", decimal.mark = ".", trim = TRUE,
             largest_with_cents = 1e+05, negative_parens = FALSE, ...)
dollar(x, accuracy = NULL, scale = 1, prefix = "$", suffix = "",
       big.mark = ",", decimal.mark = ".", trim = TRUE,
       largest_with_cents = 1e+05, negative_parens = FALSE, ...)
```

Arguments

- **accuracy**: Number to round to, `NULL` for automatic guess.
- **scale**: A scaling factor: `x` will be multiply by `scale` before formating (useful to display the data on another scale, e.g. in k$).
- **prefix, suffix**: Symbols to display before and after value.
- **big.mark**: Character used between every 3 digits to separate thousands.
- **decimal.mark**: The character to be used to indicate the numeric decimal point.
- **trim**: Logical, if `FALSE`, values are right-justified to a common width (see `base::format()`).
- **largest_with_cents**: The value that all values of `x` must be less than in order for the cents to be displayed.
- **negative_parens**: Should negative values be shown with parentheses?
- **...**: Other arguments passed on to `base::format()`.
- **x**: A numeric vector to format.
Value

A function with single parameter \( x \), a numeric vector, that returns a character vector.

Examples

dollar(%)(c(-100, 0.23, 1.456565, 2e3))
dollar%)(c(1:10 * 10))
dollar(c(100, 0.23, 1.456565, 2e3))
dollar(c(1:10 * 10))
dollar(10^1:8))

usd <- dollar%)(prefix = "USD ")
usd(c(100, -100))

euro <- dollar%)(prefix = ", suffix = " \u20ac")
euro(100)

finance <- dollar%)(negative_parens = TRUE)
finance(c(-100, 100))

dscale

Discrete scale.

Description

Discrete scale.

Usage

dscale(x, palette, na.value = NA)

Arguments

x vector of discrete values to scale
palette aesthetic palette to use
na.value aesthetic to use for missing values

Examples

with(mtcars, plot(disp, mpg, pch = 20, cex = 3,
col = dscale(factor(cyl), brewer_pal())))
**expand_range**

*Expand a range with a multiplicative or additive constant.*

**Description**

Expand a range with a multiplicative or additive constant.

**Usage**

```r
expand_range(range, mul = 0, add = 0, zero_width = 1)
```

**Arguments**

- `range`: range of data, numeric vector of length 2
- `mul`: multiplicative constant
- `add`: additive constant
- `zero_width`: distance to use if range has zero width

---

**exp_trans**

*Exponential transformation (inverse of log transformation).*

**Description**

Exponential transformation (inverse of log transformation).

**Usage**

```r
exp_trans(base = exp(1))
```

**Arguments**

- `base`: Base of logarithm
extended_breaks

Extended breaks. Uses Wilkinson’s extended breaks algorithm as implemented in the labeling package.

Description

Extended breaks. Uses Wilkinson’s extended breaks algorithm as implemented in the labeling package.

Usage

extended_breaks(n = 5, ...)

Arguments

n desired number of breaks
... other arguments passed on to labeling::extended()

References


Examples

extended_breaks()(1:10)
extended_breaks()(1:100)

format_format

Format with using any arguments to format().

Description

If the breaks have names, they will be used in preference to formatting the breaks.

Usage

format_format(...) 

Arguments

... other arguments passed on to format().

See Also

format(), format.Date(), format.POSIXct()
**gradient_n_pal**

*Arbitrary colour gradient palette (continuous).*

**Description**

Arbitrary colour gradient palette (continuous).

**Usage**

```
gradient_n_pal(colours, values = NULL, space = "Lab")
```

**Arguments**

- **colours**: vector of colours
- **values**: if colours should not be evenly positioned along the gradient this vector gives the position (between 0 and 1) for each colour in the `colours` vector. See `rescale()` for a convenience function to map an arbitrary range to between 0 and 1.
- **space**: colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

**grey_pal**

*Grey scale palette (discrete).*

**Description**

Grey scale palette (discrete).

**Usage**

```
grey_pal(start = 0.2, end = 0.8)
```

**Arguments**

- **start**: grey value at low end of palette
- **end**: grey value at high end of palette

**See Also**

`seq_gradient_pal()` for continuous version

**Examples**

```
show_col(grey_pal() (25))
show_col(grey_pal(0, 1) (25))
```
hms_trans

Transformation for times (class hms).

Description
Transformation for times (class hms).

Usage
hms_trans()

Examples
if (require("hms")) {
  hms <- round(runif(10) * 86400)
  t <- hms_trans()
  t$transform(hms)
  t$inverse(t$transform(hms))
  t$breaks(hms)
}

hue_pal

Hue palette (discrete).

Description
Hue palette (discrete).

Usage
hue_pal(h = c(0, 360) + 15, c = 100, l = 65, h.start = 0,
        direction = 1)

Arguments
h             range of hues to use, in [0, 360]
c             chroma (intensity of colour), maximum value varies depending on combination
               of hue and luminance.
l             luminance (lightness), in [0, 100]
h.start       hue to start at
direction     direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise
Examples

```r
show_col(hue_pal())(4))
show_col(hue_pal())(9))
show_col(hue_pal(l = 90)(9))
show_col(hue_pal(l = 30)(9))

show_col(hue_pal())(9))
show_col(hue_pal(direction = -1)(9))

show_col(hue_pal())(9))
show_col(hue_pal(h = c(0, 90))(9))
show_col(hue_pal(h = c(90, 180))(9))
show_col(hue_pal(h = c(180, 270))(9))
show_col(hue_pal(h = c(270, 360))(9))
```

<table>
<thead>
<tr>
<th>identity_pal</th>
<th>Identity palette.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Leaves values unchanged - useful when the data is already scaled.</td>
</tr>
<tr>
<td></td>
<td><strong>Identity transformation (do nothing).</strong></td>
</tr>
<tr>
<td>identity_trans</td>
<td>Identity transformation (do nothing).</td>
</tr>
<tr>
<td></td>
<td>Identity transformation (do nothing).</td>
</tr>
<tr>
<td></td>
<td><strong>Line type palette (discrete).</strong></td>
</tr>
<tr>
<td>linetype_pal</td>
<td>Based on a set supplied by Richard Pearson, University of Manchester</td>
</tr>
<tr>
<td></td>
<td><strong>Linetype palette (discrete).</strong></td>
</tr>
</tbody>
</table>
log1p_trans

Log plus one transformation.

Description

Log plus one transformation.

Usage

log1p_trans()

Examples

trans_range(log_trans(), 1:10)
trans_range(log1p_trans(), 0:9)

log_breaks

Log breaks (integer breaks on log-transformed scales).

Description

Log breaks (integer breaks on log-transformed scales).

Usage

log_breaks(n = 5, base = 10)

Arguments

n  desired number of breaks
base  base of logarithm to use

Examples

log_breaks(c(1, 1e6))
log_breaks(c(1, 1e5))
log_breaks(c(1664, 14008))
log_breaks(c(407, 3430))
log_breaks(c(1761, 8557))
log_trans

*Log transformation.*

**Description**

Log transformation.

**Usage**

```r
log_trans(base = exp(1))
```

**Arguments**

- `base` base of logarithm

---

manual_pal

*Manual palette (manual).*

**Description**


**Usage**

```r
manual_pal(values)
```

**Arguments**

- `values` vector of values to be used as a palette.

---

math_format

*Add arbitrary expression to a label. The symbol that will be replace by the label value is .x.*

**Description**

Add arbitrary expression to a label. The symbol that will be replace by the label value is .x.

**Usage**

```r
math_format(expr = 10^.x, format = force)
```
**muted**

Arguments

- `expr` expression to use
- `format` another format function to apply prior to mathematical transformation - this makes it easier to use floating point numbers in mathematical expressions.

Value

a function with single parameter `x`, a numeric vector, that returns a list of expressions

See Also

- `plotmath()`

Examples

```r
math_format()(1:10)
math_format(alpha + frac(1, .x))(1:10)
math_format()(runif(10))
math_format(format = percent)(runif(10))
```

---

**muted**

*Mute standard colour.*

Description

Mute standard colour.

Usage

```r
muted(colour, l = 30, c = 70)
```

Arguments

- `colour` character vector of colours to modify
- `l` new luminance
- `c` new chroma

Examples

```r
muted("red")
muted("blue")
show_col(c("red", "blue", muted("red"), muted("blue")))
```
number_bytes_format  Bytes formatter: convert to byte measurement and display symbol.

Description
Bytes formatter: convert to byte measurement and display symbol.

Usage
number_bytes_format(symbol = "auto", units = "binary", ...)
number_bytes(x, symbol = "auto", units = c("binary", "si"), ...)

Arguments
- **symbol**: byte symbol to use. If "auto" the symbol used will be determined by the maximum value of x. Valid symbols are "b", "Kb", "Mb", "Gb", "Tb", "Pb", "Eb", "Zb", and "Yb", along with their upper case equivalents and "iB" equivalents.
- **units**: which unit base to use, "binary" (1024 base) or "si" (1000 base) for ISI units.
- **...**: other arguments passed to number()
- **x**: a numeric vector to format

Value
A function with three parameters, `x`, a numeric vector that returns a character vector, `symbol` the byte symbol for ISI metric units).

References

Examples
number_bytes_format(sample(3000000000, 10))
number_bytes(sample(3000000000, 10))
number_bytes(sample(3000000000, 10), accuracy = .1)
Number formatters

Description

A set of functions to format numeric values:

- `number_format()` and `number()` are generic formatters for numbers.
- `comma_format()` and `comma()` format numbers with commas separating thousands.
- `percent_format()` and `percent()` multiply values by one hundred and display percent sign.
- `unit_format()` add units to the values.

All formatters allow you to re-scale (multiplicatively), to round to specified accuracy, to add custom suffix and prefix and to specify decimal.mark and big.mark.

Usage

```r
number_format(accuracy = 1, scale = 1, prefix = "", suffix = "", big.mark = " ", decimal.mark = ".", trim = TRUE, ...)

number(x, accuracy = 1, scale = 1, prefix = "", suffix = "", big.mark = " ", decimal.mark = ".", trim = TRUE, ...)

comma_format(accuracy = 1, scale = 1, prefix = "", suffix = "", big.mark = ",", decimal.mark = ".", trim = TRUE, ...)

comma(x, accuracy = 1, scale = 1, prefix = "", suffix = "", big.mark = ",", decimal.mark = ".", trim = TRUE, ...)

percent_format(accuracy = NULL, scale = 100, prefix = "", suffix = "\%", big.mark = ",", decimal.mark = ".", trim = TRUE, ...)

percent(x, accuracy = NULL, scale = 100, prefix = "", suffix = "\%", big.mark = ",", decimal.mark = ".", trim = TRUE, ...)

unit_format(accuracy = 1, scale = 1, prefix = "", unit = "m", sep = " ", suffix = paste0(sep, unit), big.mark = " ", decimal.mark = ".", trim = TRUE, ...)
```

Arguments

- **accuracy** Number to round to, NULL for automatic guess.
- **scale** A scaling factor: x will be multiply by scale before formatting (useful if the underlying data is on another scale, e.g. for computing percentages or thousands).
prefix, suffix  Symbols to display before and after value.
big.mark    Character used between every 3 digits to separate thousands.
decimal.mark The character to be used to indicate the numeric decimal point.
trim        Logical, if FALSE, values are right-justified to a common width (see base::format()).
...          Other arguments passed on to base::format().
x           A numeric vector to format.
digits      Deprecated, use accuracy instead.
unit        The units to append.
sep          The separator between the number and the unit label.

Value

*_format() returns a function with single parameter x, a numeric vector, that returns a character vector.

Examples

```r
# number()

v <- c(12.3, 4, 12345.789, 0.0002)
number(v)
number(v, big.mark = "","
number(v, accuracy = .001)
number(v, accuracy = .001, decimal.mark = "","
number(v, accuracy = .5)

# number_format()

my_format <- number_format(big.mark = ""," decimal.mark = "","
my_format(v)

# comma() and comma_format()

comma_format(c(1e3, 2000, 1e6))
comma_format(accuracy = .01)(c(1e3, 2000, 1e6))

# percent() and percent_format()

percent_format(runif(10))
percent(runif(10))
per_mille <- percent_format(
  scale = 1000,
  suffix = \‰,
  accuracy = .1
)
per_mille(.1234)

french_percent <- percent_format(
  decimal.mark = ",",
  suffix = " %"
)
```
french_percent(runif(10))

# unit_format()
# labels in kilometer when the raw data are in meter
km <- unit_format(unit = "km", scale = 1e-3, digits = 2)
km(runif(10) * 1e3)

# labels in hectares, raw data in square meters
ha <- unit_format(unit = "ha", scale = 1e-4)
km(ha(runif(10) * 1e5))

---

**ordinal_format**  
**Ordinal formatter: add ordinal suffixes (-st, -nd, -rd, -th) to numbers.**

**Description**

`ordinal_english()`, `ordinal_french()` and `ordinal_spanish()` provide rules for computing ordinal indicators in English, French and Spanish respectively.

**Usage**

```r
ordinal_format(prefix = "", suffix = "", big.mark = " ",
               rules = ordinal_english(), ...)

ordinal(x, prefix = "", suffix = "", big.mark = " ",
        rules = ordinal_english(), ...)

ordinal_english()

ordinal_french()

ordinal_spanish()
```

**Arguments**

- `prefix, suffix` Symbols to display before and after value.
- `big.mark` Character used between every 3 digits to separate thousands.
- `rules` Named list of regular expressions, match in order. Name gives suffix, and value specifies which numbers to match.
- `...` Other arguments passed on to `base::format()`.
- `x` A numeric vector of positive values to format.

**Value**

A function with single parameter `x`, a numeric vector, that returns a character vector.
Note

Values in x will be rounded before formating.

Examples

```r
ordinal_format()(1:10)
ordinal(1:10)

# Custom rules for French
french <- list(
er = "\^1\$",
nd = "\^2\$",
  e = "."  
)
ordinal(1:20, rules = french)

# You can also use directly
ordinal(1:20, rules = ordinal_french())
```

package-scales

Generic plot scaling methods

Description

Generic plot scaling methods

parse_format

Parse a text label to produce expressions for plotmath.

Description

Parse a text label to produce expressions for plotmath.

Usage

```r
parse_format()
```

Value

a function with single parameter x, a character vector, that returns a list of expressions

See Also

```r
plotmath()
```

Examples

```r
parse_format()(c("alpha", "beta", "gamma"))
```
**pretty_breaks**

*Pretty breaks. Uses default R break algorithm as implemented in pretty().*

Description

Pretty breaks. Uses default R break algorithm as implemented in `pretty()`.  

Usage

```
pretty_breaks(n = 5, ...)  
```

Arguments

- `n` desired number of breaks  
- `...` other arguments passed on to `pretty()`  

Examples

```
pretty_breaks()(1:10)  
pretty_breaks()(1:100)  
pretty_breaks()(as.Date(c("2008-01-01", "2009-01-01")))  
pretty_breaks()(as.Date(c("2008-01-01", "2009-01-01")))  
```

---

**probability_trans**

*Probability transformation.*

Description

Probability transformation.  

Usage

```
probability_trans(distribution, ...)  
```

Arguments

- `distribution` probability distribution. Should be standard R abbreviation so that "p" + distribution is a valid probability density function, and "q" + distribution is a valid quantile function.  
- `...` other arguments passed on to distribution and quantile functions
**pseudo_log_trans**  
*Pseudo-log transformation*

**Description**
A transformation mapping numbers to a signed logarithmic scale with a smooth transition to linear scale around 0.

**Usage**
```
pseudo_log_trans(sigma = 1, base = exp(1))
```

**Arguments**
- `sigma`: scaling factor for the linear part
- `base`: approximate logarithm base used

**pvalue_format**  
*p-values formatter*

**Description**
Formatter for p-values, adding a symbol "<" for small p-values.

**Usage**
```
pvalue_format(accuracy = 0.001, decimal.mark = ".", add_p = FALSE)
pvalue(x, accuracy = 0.001, decimal.mark = ".", add_p = FALSE)
```

**Arguments**
- `accuracy`: Number to round to.
- `decimal.mark`: The character to be used to indicate the numeric decimal point.
- `add_p`: Add "p=" before the value?
- `x`: A numeric vector of p-values.

**Value**
pvalue_format returns a function with single parameter `x`, a numeric vector, that returns a character vector.
Examples

```r
p <- c(.50, .012, .045, .011, .009, 0.00002, NA)
pvalue(p)
pvalue(p, accuracy = .01)
pvalue(p, add_p = TRUE)
custom_function <- pvalue_format(accuracy = .1, decimal.mark = "",)
custom_function(p)
```

Description

Mutable ranges have two methods (`train` and `reset`), and make it possible to build up complete ranges with multiple passes.

Usage

Range

Format

An object of class `R6ClassGenerator` of length 24.

reciprocal_trans  Reciprocal transformation.

Description

Reciprocal transformation.

Usage

```r
reciprocal_trans()
```
regular_minor_breaks  Minor breaks. Places minor breaks between major breaks.

Description

Minor breaks. Places minor breaks between major breaks.

Usage

regular_minor_breaks(reverse = FALSE)

Arguments

reverse if TRUE, calculates the minor breaks for a reversed scale

Examples

m <- extended_breaks(c(1, 10))
regular_minor_breaks(m, c(1, 10), n = 2)

n <- extended_breaks(c(0, -9))
regular_minor_breaks(reverse = TRUE)(n, c(0, -9), n = 2)

rescale  Rescale continuous vector to have specified minimum and maximum.

Description

Rescale continuous vector to have specified minimum and maximum.

Usage

rescale(x, to, from, ...)

## S3 method for class 'numeric'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)

## S3 method for class 'dist'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)

## S3 method for class 'logical'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)
## Description

Rescale numeric vector to have specified maximum.

## Usage

```r
rescale_max(x, to = c(0, 1), from = range(x, na.rm = TRUE))
```

## Arguments

- `x`: numeric vector of values to manipulate.
- `to`: output range (numeric vector of length two)
- `from`: input range (numeric vector of length two). If not given, is calculated from the range of `x`
Examples

rescale_max(1:100)
rescale_max(runif(50))
rescale_max(1)

rescale_mid Rescale vector to have specified minimum, midpoint, and maximum.

Description

Rescale vector to have specified minimum, midpoint, and maximum.

Usage

rescale_mid(x, to, from, mid, ...)

## S3 method for class 'numeric'
rescale_mid(x, to = c(0, 1), from = range(x, na.rm = TRUE), mid = 0, ...)

## S3 method for class 'logical'
rescale_mid(x, to = c(0, 1), from = range(x, na.rm = TRUE), mid = 0, ...)

## S3 method for class 'dist'
rescale_mid(x, to = c(0, 1), from = range(x, na.rm = TRUE), mid = 0, ...)

## S3 method for class 'POSIXt'
rescale_mid(x, to = c(0, 1), from = range(x, na.rm = TRUE), mid, ...)

## S3 method for class 'Date'
rescale_mid(x, to = c(0, 1), from = range(x, na.rm = TRUE), mid, ...)

## S3 method for class 'integer64'
rescale_mid(x, to = c(0, 1), from = range(x, na.rm = TRUE), mid = 0, ...)

Arguments

x vector of values to manipulate.
to output range (numeric vector of length two)
from input range (vector of length two). If not given, is calculated from the range of x
rescale_none

mid mid-point of input range
... other arguments passed on to methods

Examples
rescale_mid(1:100, mid = 50.5)
rescale_mid(runif(50), mid = 0.5)
rescale_mid(1)

rescale_none Don’t perform rescaling

Description
Don’t perform rescaling

Usage
rescale_none(x, ...)

Arguments
x numeric vector of values to manipulate.
... all other arguments ignored

Examples
rescale_none(1:100)

rescale_pal Rescale palette (continuous)

Description
Just rescales the input to the specific output range. Useful for alpha, size, and continuous position.

Usage
rescale_pal(range = c(0.1, 1))

Arguments
range Numeric vector of length two, giving range of possible values. Should be between 0 and 1.
reverse_trans  \hspace{1cm} Reverse transformation.

Description

Reverse transformation.

Usage

reverse_trans()

scientific_format  \hspace{1cm} Scientific formatter.

Description

Scientific formatter.

Usage

scientific_format(digits = 3, scale = 1, prefix = "", suffix = "",
decimal.mark = ".", trim = TRUE, ...)

scientific(x, digits = 3, scale = 1, prefix = "", suffix = "",
decimal.mark = ".", trim = TRUE, ...)

Arguments

digits  \hspace{1cm} Number of significant digits to show.
scale  \hspace{1cm} A scaling factor: x will be multiply by scale before formatting (useful if the underlying data is on another scale, e.g. for computing percentages or thousands).

prefix, suffix  \hspace{1cm} Symbols to display before and after value.
decimal.mark  \hspace{1cm} The character to be used to indicate the numeric decimal point.
trim  \hspace{1cm} Logical, if FALSE, values are right-justified to a common width (see base::format()).
...
\hspace{1cm} Other arguments passed on to base::format().
x  \hspace{1cm} A numeric vector to format.

Value

A function with single parameter x, a numeric vector, that returns a character vector.
**seq_gradient_pal**

Examples

```r
scientific_format()([1:10]
scientific_format()([runif(10))
scientific_format(digits = 2)(runif(10))
scientific([1:10)
scientific(runif(10))
scientific(runif(10), digits = 2)
scientific(12345, suffix = " cells/mL")
```

---

**seq_gradient_pal**  
Sequential colour gradient palette (continuous).

**Description**

Sequential colour gradient palette (continuous).

**Usage**

```r
seq_gradient_pal(low = mnsl("10B 4/6"), high = mnsl("10R 4/6"),
space = "Lab")
```

**Arguments**

- **low**: colour for low end of gradient.
- **high**: colour for high end of gradient.
- **space**: colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

**Examples**

```r
x <- seq(0, 1, length.out = 25)
show_col(seq_gradient_pal(x))
show_col(seq_gradient_pal("white", "black"))(x)
library(munsell)
show_col(seq_gradient_pal("white", mnsl("10R 4/6"))(x))
```
shape_pal  

**Description**

Shape palette (discrete).

**Usage**

```r
shape_pal(solid = TRUE)
```

**Arguments**

- **solid**
  - should shapes be solid or not?

show_col  

**Description**

A quick and dirty way to show colours in a plot.

**Usage**

```r
show_col(colours, labels = TRUE, borders = NULL, cex_label = 1)
```

**Arguments**

- **colours**
  - a character vector of colours
- **labels**
  - boolean, whether to show the hexadecimal representation of the colours in each tile
- **borders**
  - colour of the borders of the tiles; matches the border argument of `graphics::rect()`. The default means `par("fg")`. Use `border = NA` to omit borders.
- **cex_label**
  - size of printed labels, works the same as `cex` parameter of `plot()`

sqrt_trans  

**Description**

Square-root transformation.

**Usage**

```r
sqrt_trans()
```
Squish values into range.

**Usage**

```
squish(x, range = c(0, 1), only.finite = TRUE)
```

**Arguments**

- `x`: numeric vector of values to manipulate.
- `range`: numeric vector of length two giving desired output range.
- `only.finite`: if TRUE (the default), will only modify finite values.

**Examples**

```
squish(c(-1, 0.5, 1, 2, NA))
squish(c(-1, 0, 0.5, 1, 2))
```

Squish infinite values to range.

**Usage**

```
squish_infinite(x, range = c(0, 1))
```

**Arguments**

- `x`: numeric vector of values to manipulate.
- `range`: numeric vector of length two giving desired output range.

**Examples**

```
squish_infinite(c(-Inf, -1, 0, 1, 2, Inf))
```
time_trans  
*Transformation for date-times (class POSIXt).*

**Description**
Transformation for date-times (class POSIXt).

**Usage**
\[\text{time_trans}(tz = \text{NULL})\]

**Arguments**
- **tz**
  Optionally supply the time zone. If `NULL`, the default, the time zone will be extracted from first input with a non-null tz.

**Examples**
```
hours <- \text{seq(ISOdate(2000,3,20, tz = ",", by = "hour", length.out = 10)}
t <- \text{time_trans()}
t$\text{transform(hours)}
t$\text{inverse(t$\text{transform(hours)})}
t$\text{format(t$\text{breaks(range(hours))})}
```

---

**train_continuous**  
*Train (update) a continuous scale*

**Description**
Train (update) a continuous scale

**Usage**
\[\text{train_continuous(new, existing = NULL)}\]

**Arguments**
- **new**
  New data to add to scale
- **existing**
  Optional existing scale to update
**train_discrete**

Train (update) a discrete scale

**Description**

Train (update) a discrete scale

**Usage**

\[
\text{train_discrete(} \text{new, existing = NULL, drop = FALSE, na.rm = FALSE})
\]

**Arguments**

- **new**: New data to add to scale
- **existing**: Optional existing scale to update
- **drop**: TRUE, will drop factor levels not associated with data
- **na.rm**: If TRUE, will remove missing values

**trans_breaks**

Pretty breaks on transformed scale.

**Description**

These often do not produce very attractive breaks.

**Usage**

\[
\text{trans_breaks(} \text{trans, inv, n = 5, \ldots})
\]

**Arguments**

- **trans**: function of single variable, \(x\), that given a numeric vector returns the transformed values
- **inv**: inverse of the transformation function
- **n**: desired number of ticks
- **\ldots**: other arguments passed on to pretty

**Examples**

\[
\text{trans_breaks(} \text{"log10", function(x) 10 ^ x)(c(1, 1e6))}
\]
\[
\text{trans_breaks(} \text{"sqrt", function(x) x ^ 2)(c(1, 100))}
\]
\[
\text{trans_breaks(} \text{function(x) 1 / x, function(x) 1 / x)(c(1, 100))}
\]
\[
\text{trans_breaks(} \text{function(x) -x, function(x) -x)(c(1, 100))}
\]
trans_format

Format labels after transformation.

Description

Format labels after transformation.

Usage

trans_format(trans, format = scientific_format())

Arguments

trans transformation to apply
format additional formatter to apply after transformation

Value

a function with single parameter x, a numeric vector, that returns a character vector of list of expressions

Examples

tf <- trans_format("log10", scientific_format())
tf(10 ^ 1:6)

trans_new

Create a new transformation object.

Description

A transformation encapsulates a transformation and its inverse, as well as the information needed to create pleasing breaks and labels. The breaks function is applied on the transformed range of the range, and it’s expected that the labels function will perform some kind of inverse transformation on these breaks to give them labels that are meaningful on the original scale.

Usage

trans_new(name, transform, inverse, breaks = extended_breaks(),
minor_breaks = regular_minor_breaks(), format = format_format(),
domain = c(-Inf, Inf))
Arguments

name transformation name
transform function, or name of function, that performs the transformation
inverse function, or name of function, that performs the inverse of the transformation
breaks default breaks function for this transformation. The breaks function is applied to the raw data.
minor_breaks default minor breaks function for this transformation.
format default format for this transformation. The format is applied to breaks generated to the raw data.
domain domain, as numeric vector of length 2, over which transformation is valued

See Also

asn_trans, atanh_trans, boxcox_trans, date_trans, exp_trans, hms_trans, identity_trans,
log10_trans, log1p_trans, log2_trans, log_trans, logit_trans, modulus_trans, probability_trans,
probit_trans, pseudo_log_trans, reciprocal_trans, reverse_trans, sqrt_trans, time_trans

trans_range

Compute range of transformed values.

Description

Silently drops any ranges outside of the domain of `trans`.

Usage

`trans_range(trans, x)`

Arguments

trans a transformation object, or the name of a transformation object given as a string.
x a numeric vector to compute the range of
### viridis_pal 

**Viridis palette**

**Description**

Viridis palette

**Usage**

```r
viridis_pal(alpha = 1, begin = 0, end = 1, direction = 1, 
option = "D")
```

**Arguments**

- **alpha**
  
  The alpha transparency, a number in [0,1], see argument alpha in `hsv`.

- **begin**
  
  The (corrected) hue in [0,1] at which the viridis colormap begins.

- **end**
  
  The (corrected) hue in [0,1] at which the viridis colormap ends.

- **direction**
  
  Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed.

- **option**
  
  A character string indicating the colormap option to use. Four options are available: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E").

**References**

https://bids.github.io/colormap/

**Examples**

```r
show_col(viridis_pal()@10))
show_col(viridis_pal(direction = -1)(6))
show_col(viridis_pal(begin = 0.2, end = 0.8)(4))
show_col(viridis_pal(option = "plasma")(6))
```

---

### wrap_format

**Wrap text to a specified width, adding newlines for spaces if text exceeds the width**

**Description**

Wrap text to a specified width, adding newlines for spaces if text exceeds the width

**Usage**

```r
wrap_format(width)
```
zero_range

Arguments

width value above which to wrap

Value

Function with single parameter x, a character vector, that returns a wrapped character vector

Examples

```
wrap_10 <- wrap_format(10)
wrap_10('A long line that needs to be wrapped')
```

zero_range

Determine if range of vector is close to zero, with a specified tolerance

Description

The machine epsilon is the difference between 1.0 and the next number that can be represented by the machine. By default, this function uses epsilon * 1000 as the tolerance. First it scales the values so that they have a mean of 1, and then it checks if the difference between them is larger than the tolerance.

Usage

```
zero_range(x, tol = 1000 * .Machine$double.eps)
```

Arguments

x numeric range: vector of length 2
tol A value specifying the tolerance.

Value

logical TRUE if the relative difference of the endpoints of the range are not distinguishable from 0.

Examples

```
eps <- .Machine$double.eps
zero_range(c(1, 1 + eps)) # TRUE
zero_range(c(1, 1 + 99 * eps)) # TRUE
zero_range(c(1, 1 + 1001 * eps)) # FALSE - Crossed the tol threshold
zero_range(c(1, 1 + 2 * eps), tol = eps) # FALSE - Changed tol

# Scaling up or down all the values has no effect since the values
# are rescaled to 1 before checking against tol
zero_range(100000 * c(1, 1 + eps)) # TRUE
zero_range(100000 * c(1, 1 + 1001 * eps)) # FALSE
zero_range(.00001 * c(1, 1 + eps)) # TRUE
```
```r
zero_range(.00001 * c(1, 1 + 1001 * eps)) # FALSE

# NA values
zero_range(c(1, NA)) # NA
zero_range(c(1, NaN)) # NA

# Infinite values
zero_range(c(1, Inf)) # FALSE
zero_range(c(-Inf, Inf)) # FALSE
zero_range(c(Inf, Inf)) # TRUE
```
Index

+Topic datasets
  Range, 33
+Topic manip
  rescale, 34

abs_area, 3
alpha, 4
area_pal, 4, 12
as.trans, 5
asn_trans, 5, 12, 45
atanh_trans, 5, 12, 45
base::cut(), 11
base::format(), 16, 28, 29, 38
boxcox_trans, 6, 12, 45
brewer_pal, 7, 12
cbreaks, 7
censor, 8
col2hcl, 9
col_bin (col_numeric), 10
col_factor (col_numeric), 10
col_numeric, 10
col_quartile (col_numeric), 10
colorRamp, 9, 10
colour_ramp, 9
comma (number_format), 27
comma_format (number_format), 27
ContinuousRange (Range), 33
cscale, 12
date_breaks, 13
date_format, 13
date_trans, 12, 14, 45
dichromat_pal, 12, 14
discard, 15
DiscreteRange (Range), 33
div_gradient_pal, 12, 15
dollar (dollar_format), 16
dollar_format, 16

dscale, 17
exp_trans, 12, 18, 45
expand_range, 18
extended_breaks, 19
format(), 19
format.Date(), 19
format.POSIXct(), 19
format_format, 19
gradient_n_pal, 12, 20
graphics::rect(), 40
grDevices::col2rgb(), 9
grDevices::colors(), 9
grDevices::palette(), 9
grey_pal, 12, 20
hms_trans, 12, 21, 45
hsv, 46
hue_pal, 12, 21
identity_pal, 12, 22
identity_trans, 12, 22, 45
is_trans (trans_new), 44
labeling::extended(), 19
linetype_pal, 12, 22
log10_trans, 12, 45
log10_trans (log_trans), 24
log1p_trans, 12, 23, 45
log2_trans, 12, 45
log2_trans (log_trans), 24
log_breaks, 23
log_trans, 12, 24, 45
logit_trans, 12, 45
logit_trans (probability_trans), 31
manual_pal, 12, 24
math_format, 24
modulus_format, 24
moduli_format, 24
modulus_trans, 12, 45
modulus_trans (boxcox_trans), 6
muted, 25

number (number_format), 27
number(), 26
number_bytes (number_bytes_format), 26
number_bytes_format, 26
number_format, 27

ordinal (ordinal_format), 29
ordinal_english (ordinal_format), 29
ordinal_format, 29
ordinal_french (ordinal_format), 29
ordinal_spanish (ordinal_format), 29

packageMscales, 30
packageMscalesMpackage (packageMscales), 30
parse_format, 30
percent (number_format), 27
percent_format (number_format), 27
plotmath(), 25, 30
prettify(), 11, 31
prettify_breaks, 31
probability_trans, 12, 31, 45
probit_trans, 12, 45
probit_trans (probability_trans), 31
pseudo_log_trans, 12, 32, 45
pvalue (pvalue_format), 32
pvalue_format, 32

Range, 33
RCColorBrewer::brewer_pal(), 7
reciprocal_trans, 12, 33, 45
regular_minor_breaks, 34
rescale, 34
rescale(), 20
rescale_max, 35
rescale_mid, 36
rescale_none, 37
rescale_pal, 12, 37
reverse_trans, 12, 38, 45

scales (package-scales), 30
scientific (scientific_format), 38
scientific_format, 38
seq(), 11
seq_gradient_pal, 12, 39
seq_gradient_pal(), 20

shape_pal, 12, 40
show_col, 40
sqrt_trans, 12, 40, 45
squish, 41
squish_infinite, 41
stats::quantile(), 11
strptime(), 13

time_format (date_format), 13
time_trans, 12, 42, 45
timezones(), 13
train_continuous, 42
train_discrete, 43
trans (trans_new), 44
trans_breaks, 43
trans_format, 44
trans_new, 44
trans_range, 45

unit_format (number_format), 27
viridis_pal, 12, 46
wrap_format, 46
zero_range, 47