Package 'roperators'

June 3, 2026

Title Additional Operators to Help You Write Cleaner R Code

Version 1.4.0

Description A set of additional operators and helper functions to make R code easier to read, write, and maintain. Includes string arithmetic (such as 'foo' + 'bar'), in-place reassignment operators (such as x += 1), logical operators that handle missing values, floating-point and strict ('JavaScript'-style) equality tests, 'between' operators, and 'SQL'-style pattern matching. Also provides convenience helpers for type conversion, operating-system checks, complete-cases statistics, and string manipulation, such as Oxford-comma pasting and extracting the first, last, n-th, or most common element of a vector or word in a string. The goal is to give R users, particularly those coming from other languages such as 'Python', a friendlier and more consistent syntax.

License MIT + file LICENSE

Copyright Korn Ferry International

URL https://benwiseman.github.io/roperators/, https://github.com/BenWiseman/roperators

BugReports https://github.com/BenWiseman/roperators/issues

Depends R (>= 3.0.0)

Imports stats, tools

Suggests magrittr, knitr, markdown, rmarkdown, prettydoc, rvest, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Collate 'complete_cases.R' 'content_checks.R' 'file_checks.R' 'ip_checks.R' 'type_checks.R' 'operators.R' 'os_checks.R' 'paste_functions.R' 'shorthand.R' 'sugar.R' 'utils.R'

NeedsCompilation no

Author Ben Wiseman [cre, aut, ccp],
Steven Nydick [aut, ccp] (ORCID:
<https://orcid.org/0000-0002-2908-1188>),
Jeff Jones [aut, led]

Maintainer Ben Wiseman <benjamin.h.wiseman@gmail.com>

Repository CRAN

Date/Publication 2026-06-03 11:20:02 UTC

Contents

`arithmetic_sugar` *Arithmetic convenience operators*

Description

`%+-%` builds a tolerance interval: `x %+-% y` returns `c(x - y, x + y)`, which slots straight into the "between" operators in `comparisons`. `%/0%` is a safe division that returns `NA` instead of `Inf`/`NaN` when dividing by zero, so a stray zero will not poison a downstream sum or mean.

Usage

```
x %+-% y

x %/0% y
```

Arguments

x	a numeric value (or vector)
y	a numeric value (or vector)

Value

`%+-%` returns a length-2 numeric vector `c(lower, upper)`; `%/0%` returns `x / y` with any divide-by-zero results replaced by `NA`.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
5 %+-% 0.5             # 4.5 5.5
4.9 %><% (5 %+-% 0.5)  # TRUE  - composes with the 'between' operator

10 %/0% 2                    # 5
10 %/0% 0                    # NA (not Inf)
c(1, 2, 3) %/0% c(1, 0, 3)   # 1 NA 1
```

as.percent *Format a proportion as a percentage string*

Description

Turn a proportion (such as `0.75`) into a human-friendly percentage string (such as `"75%"`).

Usage

```
as.percent(x, digits = 1, ...)
```

Arguments

`x`	a numeric proportion, or vector of proportions, where `1` represents one hundred percent
`digits`	number of decimal places to show
`...`	further arguments passed to `formatC`

Value

A character vector of percentage strings.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
as.percent(0.75)              # "75.0\%"
as.percent(c(0.1, 0.005))     # "10.0\%" "0.5\%"
as.percent(2 / 3, digits = 0) # "67\%"
```

assign_ops *Assignment (in-place modifier) operators*

Description

Modify the stored value of the left-hand-side object in place. These are the equivalent of operators such as `+=`, `-=`, `*=`, and `/=` in languages like C++ or 'Python'. `%+=%` and `%-=%` also work with strings, and `%-=%` accepts regular expressions.

Usage

```
x %+=% y

x %-=% y

x %*=% y

x %/=% y

x %^=% y

x %log=% y

x %root=% y
```

Arguments

x	a stored value
y	value to modify the stored value by

Value

Used for the side effect of reassigning x in the calling environment; returns the new value of x invisibly.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
x <- 1
x %+=% 2
x == 3 # TRUE

x %-=% 3
x == 0 # TRUE

# Or with data frames...
test <- iris

# Simply modify in place
test$Sepal.Length[test$Species == 'setosa' & test$Petal.Length < 1.5] %+=% 1

# Which is much nicer than typing:
test$Sepal.Length[test$Species == 'setosa' & test$Petal.Length < 1.5] <-
test$Sepal.Length[test$Species == 'setosa' & test$Petal.Length < 1.5] + 1
# ...which is over the 100 character limit for R documentation!

# %+=% and %-=% also work with strings
```

```
x <- "ab"
x %+=% "c"
x %-=% "b"
x == "ac" # TRUE

# %-=% can also take regular expressions
x <- "foobar"
x %-=% "[fb]"
print(x)
# "ooar"
```

---

choose_permute *Choose and permute operators*

Description

Shorthand infix operators for common combinatorics: `n %C% k` gives the number of combinations ("n choose k"), and `n %P% k` gives the number of permutations ("n permute k").

Usage

```
n %C% k

n %P% k
```

Arguments

n	whole number (the n in "n choose/permute k")
k	whole number (the k in "n choose/permute k")

Value

A numeric value.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
5 %C% 3 # 10  (5 choose 3)
5 %P% 3 # 60  (5 permute 3)
```

chr *Cleaner type-conversion functions*

Description

Short aliases for the most common `as.*` conversions. There is nothing magical here, but the shorter names can make data-wrangling code much easier to read, especially for users coming from other languages. `as.class()` additionally converts to a class chosen by name at run time.

Usage

```
chr(x, ...)

int(x, ...)

dbl(x, ...)

num(x, ...)

bool(x, ...)

as.class(x, class)
```

Arguments

`x`	value to be converted
`...`	further arguments passed to the underlying `as.*` function
`class`	character; the name of the class to convert `x` to (used by `as.class`)

Value

The value of `x` coerced to the requested type.

Author(s)

Steven Nydick, <steven.nydick@kornferry.com>

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
chr(42)    # "42" = as.character()
int(42.1)  # 42L  = as.integer()
dbl("42")  # 42   = as.double()
num("42")  # 42   = as.numeric()
bool(42)   # TRUE = as.logical()
```

```
# as.class() converts to an arbitrary class chosen by name:
as.class(255, "roman") # CCLV
```

`comparisons` *Comparison operators with better missing-value handling*

Description

A set of comparison operators that improve on base R by treating missing values as comparable (so two `NA`s are considered equal) and by adding convenient interval ("between") tests. The operators are:

- `%==%` - equality that treats `NA == NA` as `TRUE`.
- `%===%` - strict equality of both value *and* class, for those familiar with 'JavaScript' `===`.
- `%>=%`, `%<=%` - greater/less than or equal to, with missing-value equality.
- `%><%`, `%>=<%` - between, with the ends excluded or included respectively.

For approximate (floating-point) comparisons, see `floating_point_comparisons`.

Usage

```
x %==% y

x %===% y

x %>=% y

x %<=% y

x %><% y

x %>=<% y
```

Arguments

`x`	a vector
`y`	a vector (for the "between" operators, a length-2 vector of the form `c(lower, upper)`)

Value

A logical vector.

Author(s)

Ben Wiseman, `<benjamin.h.wiseman@gmail.com>`

See Also

Other comparisons: floating_point_comparisons

Examples

```
## Equality and ordering, with missing-value equality
c(1, NA, 3, 4)  ==  c(1, NA, 4, 3)
#  TRUE    NA  FALSE FALSE

c(1, NA, 3, 4) %==% c(1, NA, 4, 3)
#  TRUE  TRUE  FALSE FALSE

c(1, NA, 3, 4) %>=% c(1, NA, 4, 3)
#  TRUE  TRUE FALSE  TRUE

c(1, NA, 3, 4) %<=% c(1, NA, 4, 3)
#  TRUE  TRUE  TRUE FALSE

## Strict equality - a la 'JavaScript' ===
# Only TRUE if the class AND value of x and y are the same
x <- int(2)
y <- 2
x == y         # TRUE
x %===% y      # FALSE
x %===% int(y) # TRUE

## Between
# ends excluded
2 %><% c(1, 3)  # TRUE
3 %><% c(1, 3)  # FALSE

# ends included
2 %>=<% c(1, 3) # TRUE
3 %>=<% c(1, 3) # TRUE
```

---

complete_cases *Statistics/Summaries with (Only) Missing Data Removed*

Description

Univariate and bivariate summaries and statistics with the least missing data removed (such as complete-cases correlations). These are typically default arguments to standard statistics functions.

Usage

```
length_cc(x, ...)

n_unique_cc(x, ...)
```

```
min_cc(x, ...)

max_cc(x, ...)

range_cc(x, ...)

all_cc(x, ...)

any_cc(x, ...)

sum_cc(x, ...)

prod_cc(x, ...)

mean_cc(x, ...)

median_cc(x, ...)

var_cc(x, y = NULL, ...)

cov_cc(x, y = NULL, ...)

cor_cc(x, y = NULL, ...)

sd_cc(x, ...)

weighted.mean_cc(x, w, ...)

quantile_cc(x, ...)

IQR_cc(x, ...)

mad_cc(x, ...)

rowSums_cc(x, ...)

colSums_cc(x, ...)

rowMeans_cc(x, ..., rescale = FALSE)

colMeans_cc(x, ..., rescale = FALSE)
```

Arguments

x	an R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.

...	arguments to pass to wrapped functions
y	NULL (default) or a vector, matrix or data frame with compatible dimensions to x. The default is equivalent to y = x (but more efficient).
w	a numerical vector of weights the same length as x giving the weights to use for elements of x.
rescale	whether to rescale the matrix/df/vector before calculating summaries

Value

The same value as the base/stats function each one wraps (for example a numeric summary, vector, or matrix), but computed with missing values removed by default.

Examples

```
n_o <- 20
n_m <- round(n_o / 3)
x   <- rnorm(n_o)
y   <- rnorm(n_o)

x[sample(n_o, n_m)] <- NA
y[sample(n_o, n_m)] <- NA

mean_cc(x)   # mean of complete cases
mean_cc(y)
var_cc(x)    # variance of complete cases
var_cc(y)
cor_cc(x, y) # correlation between available cases

# the row/column helpers also drop NAs by default
m <- matrix(c(1, NA, 3, 4, 5, 9), nrow = 2)
rowMeans_cc(m)
colSums_cc(m)

# colMeans_cc()/rowMeans_cc() can z-score each column first via rescale = TRUE
colMeans_cc(matrix(1:6, nrow = 3), rescale = TRUE)
```

---

content_checks *Vector content checks*

Description

Quick checks for what a vector contains. `is.constant()` is TRUE when x holds at most one unique value (ignoring NA), and `is.binary()` is TRUE when it holds at most two.

Usage

```
is.constant(x)

is.binary(x)
```

Arguments

x	object to be tested

Value

A logical value.

Examples

```
is.constant(c(1, 1, 1))     # TRUE
is.constant(c(1, 2, 1))     # FALSE
is.binary(c("a", "b", NA))  # TRUE
is.binary(c("a", "b", "c")) # FALSE
```

---

f *String interpolation (f-strings for R)*

Description

Interpolate R expressions into a string, like 'Python' f-strings. Anything inside curly braces is evaluated in the calling environment and inserted into the string. Doubled braces are treated as a single literal brace.

Usage

```
f(..., .envir = parent.frame())
```

Arguments

...	one or more strings containing {expr} placeholders
.envir	the environment in which to evaluate the placeholders (defaults to the calling environment)

Value

A character vector the same length as the input, with every {expr} replaced by its evaluated, comma-collapsed value.

Note

f is also a popular name for throwaway functions, so be aware it may mask (or be masked by) a local f of your own.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
name <- "Ben"
n    <- 2
f("Hi {name}, you have {n} new messages")
f("{n} + {n} = {n + n}")

# vectors are collapsed with ", "
f("today's letters: {head(LETTERS, n)}")
```

`f.as.numeric` *Convert a factor with numeric labels into a numeric vector*

Description

Converting a factor with `as.numeric()` returns the underlying integer codes, not the labels, which is rarely what you want when the labels are themselves numbers. `f.as.numeric()` returns the labels as numbers instead.

Usage

```
f.as.numeric(x)
```

Arguments

x	a factor with numeric labels

Value

A numeric vector of the factor's labels.

Author(s)

Ulrike Grömping, <groemping@beuth-hochschule.de>

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
x <- factor(c(11, 22, 33, 99))

as.numeric(x)
# 1 2 3 4      # the integer codes - NOT usually what you want

f.as.numeric(x)
# 11 22 33 99  # the labels as numbers - usually what you want

# equivalent to the clunkier base idiom:
as.numeric(as.character(x))
```

file_checks *File Extension Checks*

Description

Check whether file extension is as specified

Usage

```
is_txt_file(x)

is_csv_file(x)

is_excel_file(x)

is_r_file(x)

is_rdata_file(x)

is_rda_file(x)

is_rds_file(x)

is_spss_file(x)

check_ext_against(x, ext = "txt")
```

Arguments

x	file(s) to be tested
ext	extension to test against

Value

a logical value

Note

These only check the file extension and not the contents of the file. Checking the contents of a file might come later but would be quite a bit more involved. You can use ‘readr‘ or ‘readxl‘ (for example) to check the file contents.

Examples

```
# create your own file extension checks
is_word_file <- function(x){
  check_ext_against(x, ext = c("doc", "docx"))
}
```

```
is_word_file(c("blah.doc", "blah.docx", "blah.txt"))
```

`floating_point_comparisons` *Floating-point comparison operators*

Description

An important set of operators missing from base R. Using == on two non-integer numbers can give unexpected results (see examples), because of the way floating-point numbers are represented. These operators instead test equality up to a small tolerance, via `all.equal`.

For a fuller explanation, see https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html.

Usage

```
x %~=% y

x %>~% y

x %<~% y
```

Arguments

x	numeric
y	numeric

Value

A logical value.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

See Also

Other comparisons: `comparisons`

Examples

```
## Floating-point test of equality
# base R:
(0.1 + 0.1 + 0.1) == 0.3   # FALSE
# with roperators:
(0.1 + 0.1 + 0.1) %~=% 0.3 # TRUE

# Note how the base >= and <= behave here:
```

```
(0.1 + 0.1 + 0.1) %>=% 0.3 # TRUE
(0.1 + 0.1 + 0.1) %<=% 0.3 # FALSE

# Use %>~% and %<~% for greater/less than OR approximately equal
(0.1 + 0.1 + 0.1) %>~% 0.3 # TRUE
(0.1 + 0.1 + 0.1) %<~% 0.3 # TRUE
```

---

get_1st *Get the first, last, n-th, or most frequent element or word*

Description

Small helpers for pulling pieces out of vectors and strings - handy inside apply-style calls. `get_1st()`, `get_last()`, and `get_nth()` extract elements of a vector; the `*_word` variants split strings into words first; and `get_most_frequent()` / `get_most_frequent_word()` return the most common value(s).

Usage

```
get_1st(x, type = "v")

get_last(x, type = "v")

get_nth(x, n = 1, type = "v")

get_1st_word(x, type = "v", split = " ")

get_last_word(x, type = "v", split = " ")

get_nth_word(x, n = 1, type = "v", split = " ")

get_most_frequent(x, collapse = NULL)

get_most_frequent_word(
  x,
  ignore.punct = TRUE,
  ignore.case = TRUE,
  split = " ",
  collapse = NULL,
  punct.regex = "[[:punct:]]",
  punct.replace = ""
)
```

Arguments

x	an R object, usually a vector or character string

type	'v' (default) to index as a vector (x[1]), or 'l' to index as a list (x[[1]])
n	integer; the n-th element or word to select
split	character used to separate words (default ' ')
collapse	optional character; if supplied, the result is pasted into a single string using this separator
ignore.punct	logical; ignore punctuation marks
ignore.case	logical; ignore case (if TRUE, the result is lower-case)
punct.regex	character; regex used to strip punctuation (default [[:punct:]])
punct.replace	character; what to replace punctuation with (default "")

Value

The selected element(s) or word(s). get_most_frequent*() return the most common value(s), as a character vector unless x is numeric and collapse is not used.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
# a list of split-up car names
car_names <- strsplit(row.names(mtcars)[1:5], " ")

sapply(car_names, get_1st)
# [1] "Mazda" "Mazda" "Datsun" "Hornet" "Hornet"

sapply(car_names, get_nth, 2)
# [1] "RX4" "RX4" "710" "4" "Sportabout"

# Or pull a simple string apart (e.g. someone's full name):
get_1st_word(rownames(mtcars)[1:5])
# [1] "Mazda" "Mazda" "Datsun" "Hornet" "Hornet"

get_last_word(rownames(mtcars)[1:5])
# [1] "RX4" "Wag" "710" "Drive" "Sportabout"

get_nth_word(rownames(mtcars)[1:5], 2)
# [1] "RX4" "RX4" "710" "4" "Sportabout"


# get_most_frequent() returns the mode(s)
my_stuff <- c(1:10, 10, 5)
get_1st(my_stuff)           # 1
get_nth(my_stuff, 3)        # 3
get_last(my_stuff)          # 5
get_most_frequent(my_stuff) # the modes (5 and 10), as a numeric vector

my_chars <- c("a", "b", "b", "a", "g", "o", "l", "d")
```

```
get_most_frequent(my_chars)                    # "a" "b"
get_most_frequent(my_chars, collapse = " & ")  # "a & b"

# the *_word helpers split a string into words first
generic_string <- "Who's A good boy? Winston's a good boy!"
get_1st_word(generic_string)    # "Who's"
get_nth_word(generic_string, 3) # "good"
get_last_word(generic_string)   # "boy!"

# default ignores case and punctuation
get_most_frequent_word(generic_string)
# keep case and punctuation:
get_most_frequent_word(generic_string, ignore.case = FALSE, ignore.punct = FALSE)
```

`integrate` *Inline integration operator*

Description

An inline call to `integrate` that returns the value of the integral directly, rather than the usual list.

Usage

```
f %integrate% range
```

Arguments

`f`	a function with a numeric return value
`range`	a length-2 numeric vector, `c(lower, upper)`

Value

A single numeric value: the value of the integral.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
f <- function(x) x^2
f %integrate% c(0, 1) # 0.3333333

# compared with base R, which returns a list:
str(integrate(f, 0, 1))
```

library.force *Load a package, installing it first if necessary*

Description

Attempts to load pkg; if it is not installed, installs it (from CRAN by default) and then loads it. `require.force` is an alias for `library.force`.

Usage

```
library.force(pkg, ...)

require.force(pkg, ...)
```

Arguments

pkg	name of the package to load or install
...	further arguments passed to `install.packages`

Value

Invisibly returns `NULL`; called for the side effect of loading (and possibly installing) pkg.

logicals *Logical operators*

Description

A few convenience logical operators: "not in" (`%ni%`), exclusive or (`%xor%`), and all-or-nothing (`%aon%`, which is `TRUE` when `x` and `y` are both `TRUE` or both `FALSE`).

Usage

```
x %ni% y

x %xor% y

x %aon% y
```

Arguments

x	a vector
y	a vector

Value

A logical vector.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
#### Not in ####
"z" %ni% c("a", "b", "c") # TRUE

#### Exclusive or ####
TRUE  %xor% TRUE  # FALSE
FALSE %xor% FALSE # FALSE
FALSE %xor% TRUE  # TRUE

#### All-or-nothing ####
TRUE  %aon% TRUE  # TRUE
FALSE %aon% FALSE # TRUE
FALSE %aon% TRUE  # FALSE
```

---

n_unique *Count the number of unique values*

Description

Returns the number of unique elements in x, optionally ignoring NAs.

Usage

```
n_unique(x, na.rm = FALSE)
```

Arguments

x	a vector
na.rm	logical; if TRUE, NAs are ignored when counting unique values

Value

An integer count of unique values.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
n_unique(c(1, 2, 2, 3, NA))              # 4
n_unique(c(1, 2, 2, 3, NA), na.rm = TRUE) # 3
```

---

os *Operating system and environment checks*

Description

Determine the current operating system and R environment, and provide simple flags for common questions such as "are we on a Mac?", "is this 64-bit R?", or "are we running inside RStudio?". These are useful when writing code that must behave differently across platforms (for example, choosing a parallel back-end on Unix versus Windows).

Usage

```
get_os()

get_R_version()

get_R_version_age(units = c("years", "months", "weeks", "days"), rounding = 2)

get_latest_CRAN_version()

get_system_python()

is.os_mac()

is.os_win()

is.os_lnx()

is.os_unx()

is.os_x64()

is.os_arm()

is.R_x64()

is.R_revo()

is.RStudio()

is.http_available()
```

Arguments

units	character; the unit to report the R version age in, one of "years", "months", "weeks", or "days".
rounding	integer; the number of decimal places to round the age to.

Value

For the is.* checks, a single logical value. get_os() returns a character string ("win", "mac", "linux", or "unix"); get_R_version() and get_latest_CRAN_version() return version strings; and get_R_version_age() returns a numeric age.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Steven Nydick, <steven.nydick@kornferry.com>

Examples

```
# determine the operating system
get_os()

# test for a particular operating system
is.os_mac()
is.os_win()
is.os_lnx()
is.os_unx()

# environment checks
is.os_x64()
is.RStudio()
get_R_version()
```

---

paste_and_cat *Paste and cat helpers*

Description

A small family of paste/cat conveniences:

- paste_() - like paste0(), but separates with an underscore.
- cat0() - like paste0(), but for cat (no separator).
- catN() - like cat0(), but appends a new line.
- paste_series() - paste a series of items together with a conjunction, e.g. "a, b, and c".
- paste_oxford() - a shortcut for paste_series() using an Oxford comma.

Usage

```
paste_(..., collapse = NULL)

cat0(..., file = "", fill = FALSE, labels = NULL, append = FALSE)

catN(..., file = "", fill = FALSE, labels = NULL, append = FALSE)

paste_series(
  ...,
  sep = c(",", ";"),
  conjunction = c("and", "or", "&"),
  use_oxford_comma = TRUE
)

paste_oxford(...)
```

Arguments

`...`	one or more R objects, to be converted to character vectors.
`collapse`	an optional character string to separate the results. Not `NA_character_`. When `collapse` is a string, the result is always a string (`character` of length 1).
`file`	character - A connection, or a character string naming the file to print to. If "" (the default), cat prints to the standard output connection, the console unless redirected by sink.
`fill`	a logical or (positive) numeric controlling how the output is broken into successive lines. see '?cat'
`labels`	character vector of labels for the lines printed. Ignored if fill is FALSE.
`append`	logical. Only used if the argument `file` is the name of file (and not a connection or `"|cmd"`). If `TRUE` output will be appended to `file`; otherwise, it will overwrite the contents of `file`.
`sep`	a character vector of strings to append after each element
`conjunction`	the conjunction used to join the final elements of a series, such as `"and"`, `"or"`, or `"&"`
`use_oxford_comma`	logical; whether to use the Oxford comma (standard in American English) before the conjunction

Value

`paste_()`, `paste_series()`, and `paste_oxford()` return a character vector. `cat0()` and `catN()` are called for their side effect (printing) and return `NULL` invisibly.

Author(s)

Steven Nydick, <steven.nydick@kornferry.com>

Examples

```
paste_("a", "b", "c")          # "a_b_c"  (paste0 with an underscore)
cat0("no", "spaces", "here")   # prints: nospaceshere
catN("...and this one", " ends with a newline")


paste_series("a")
paste_series("a", "b")
paste_series("a", "b", "c")
# works if putting entries into c function
paste_series(c("a", "b", "c"), "d")
# can use oxford comma or not
paste_series("a", "b", "c",
             use_oxford_comma = TRUE)
paste_series("a", "b", "c",
             use_oxford_comma = FALSE)
# makes no difference if fewer than 3 items
paste_series("a", "b",
             use_oxford_comma = TRUE)
```

`pattern_matching` *SQL-style pattern-matching operators*

Description

Convenience operators for regular-expression matching, inspired by SQL's `LIKE`. Each takes a character vector `x` and a single pattern, and returns a logical vector the same length as `x`.

`%rlike%` matches case-insensitively, equivalent to `grepl(pattern, x, ignore.case = TRUE)`.

`%perl%` matches case-sensitively using Perl-compatible regular expressions, equivalent to `grepl(pattern, x, perl = TRUE)`.

Usage

```
x %rlike% pattern

x %perl% pattern
```

Arguments

`x`	a character vector
`pattern`	a single character expression (regular expression)

Value

A logical vector the same length as `x`.

Note

If you are working with `data.table`, prefer its own (faster) `%like%` operator.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
x <- c("foo", "bar", "dOe", "rei", "mei", "obo")

# case-insensitive: where x contains an "o" (any case)
x[x %rlike% "O"]
# [1] "foo" "dOe" "obo"

# case-sensitive Perl matching: middle letter is an upper-case "O"
x[x %perl% "[a-z]O[a-z]"]
# [1] "dOe"
```

---

read.tsv *Read tab- or pipe-separated files*

Description

Convenience wrappers around `read.table` for tab-separated (`read.tsv`) and pipe-separated (`read.psv`) files. Both default to `header = TRUE`, like `read.csv`.

Usage

```
read.tsv(file, ...)

read.psv(file, ...)
```

Arguments

`file`	path of the file to load
`...`	further arguments passed to `read.table`

Value

A `data.frame`.

seq_around *Sequence of evenly spaced points around an origin*

Description

Returns a vector of n points evenly spaced around origin, with the given spacing between neighbours.

Usage

```
seq_around(origin = 1, n = 1, spacing = 0.25)
```

Arguments

origin	number to centre the sequence on
n	number of points to create (a single whole number)
spacing	distance between any two neighbouring points

Value

A numeric vector. Defaults to 1 when called with no arguments, to mirror the default behaviour of seq.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
seq_around(0, n = 5, spacing = 1) # -2 -1  0  1  2
seq_around(10, n = 3)             # 9.75 10.00 10.25
```

string_arithmetic *String arithmetic operators*

Description

Perform string concatenation and arithmetic in a similar way to other languages. String addition (%+%) glues strings together, string subtraction (%-%) removes a pattern, string multiplication (%s*%) repeats a string, and string division (%s/%) counts how many times a pattern occurs. String division has no equivalent in languages like 'Python', yet is arguably more useful than string multiplication, and it accepts regular expressions.

Usage

```
x %+% y

x %-% y

x %s*% y

x %s/% y
```

Arguments

x	a string (character vector)
y	a string (character vector or regular expression)

Value

A character vector for %+%, %-%, and %s*%, and an integer count for %s/%.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
("ab" %+% "c") == "abc" # TRUE
("abc" %-% "b") == "ac" # TRUE
("ac" %s*% 2) == "acac" # TRUE
("acac" %s/% "c") == 2  # TRUE
# String division with a regular expression:
"an apple a day keeps the malignant spirit of Steve Jobs at bay" %s/% "Steve Jobs|apple"
```

---

type_checks *Type checks*

Description

A collection of convenience checks for the type or contents of an object. Each returns a logical value (or vector), which keeps guard clauses and `if` statements short and readable. The `*_or_null` variants are handy for validating optional arguments, and the `*_for_calcs` / `*_for_indexing` helpers flag values that would break arithmetic or subsetting (such as `NA`, `NaN`, or `Inf`).

Usage

```
is.scalar(x)

is.scalar_or_null(x)

is.numeric_or_null(x)

is.character_or_null(x)

is.logical_or_null(x)

is.df_or_null(x)

is.list_or_null(x)

is.atomic_nan(x)

is.irregular_list(x)

any_bad_for_calcs(x, ..., na.rm = FALSE)

all_good_for_calcs(x, ..., na.rm = FALSE)

is.bad_for_indexing(x)

is.good_for_indexing(x)

is.bad_and_equal(x, y)

is.bad_for_calcs(x, na.rm = FALSE)

is.good_for_calcs(x, na.rm = FALSE)

is.null_or_na(x)
```

Arguments

x	object to be tested
...	values to be tested
na.rm	if TRUE, NA values are not considered bad for calculations
y	object to be tested

Value

A logical value (or vector) indicating whether x meets the test.

Author(s)

Steven Nydick, <steven.nydick@kornferry.com>

Examples

```
is.scalar(1)              # TRUE
is.scalar(c(1, 2))        # FALSE
is.numeric_or_null(NULL)  # TRUE
is.bad_for_calcs(NA)      # TRUE
is.good_for_calcs(1)      # TRUE
is.bad_and_equal(NA, NA)  # TRUE
```

%regex<-% *Assign to a vector only where a regular expression matches*

Description

This takes two arguments, just like gsub: a pattern and a replacement. It overwrites the *entire* element wherever the pattern matches. If you want to substitute only the matched portion, use %regex=% instead; to replace matches with nothing (""), use %-% or %-=%.

Usage

```
x %regex<-% value
```

Arguments

x	a character vector
value	a length-2 character vector of the form c(pattern, replacement)

Value

Used for the side effect of reassigning x in the calling environment; returns the modified x invisibly.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
x <- c("a1b", "b1", "c", "d0")

# overwrite any element containing a digit
x %regex<-% c("\\d+", "x")
print(x)
# "x" "b" "c" "x"
```

%~% *Fuzzy (case- and whitespace-insensitive) string equality*

Description

The string counterpart to the numeric operator %~=% (see floating_point_comparisons). x %~% y is TRUE when x and y are equal ignoring case, leading/trailing whitespace, and runs of internal whitespace - handy for joining or matching messy, hand-entered data.

Usage

```
x %~% y
```

Arguments

x	a character vector
y	a character vector

Value

A logical vector, with NA where either side is NA.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
"Foo " %~% "foo"                   # TRUE
"a  b" %~% "a b"                    # TRUE
c("Yes", "NO") %~% c("yes", "no")  # TRUE TRUE
```

%else% *Inline fallback for expressions that might error*

Description

Evaluate the left-hand side; if it raises an error, return the right-hand side instead. The right-hand side is only evaluated when needed (lazily), so an expensive or side-effecting fallback is safe to use.

Usage

```
expr %else% alternative
```

Arguments

expr	an expression to try
alternative	value (or expression) to return if expr errors

Value

The value of expr, or of alternative if expr raised an error.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
sqrt(4)         %else% NA_real_     # 2
sqrt("a")       %else% NA_real_     # NA, instead of an error
(1:3)[[99]]     %else% "out of range"
stop("boom")    %else% "recovered"
```

%regex=% *Modify an object in place by regular expression*

Description

This takes two arguments, just like gsub: a pattern and a replacement. It overwrites only the matched portion of each element of x. If you want to overwrite whole elements that match (rather than just the matched portion), use %regex<-% instead.

Usage

```
x %regex=% value
```

Arguments

x	a character vector
value	a length-2 character vector of the form c(pattern, replacement)

Value

Used for the side effect of reassigning x in the calling environment; returns the modified x invisibly.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
x <- c("a1b", "b1", "c", "d0")

# change any digit to "x"
x %regex=% c("\\d+", "x")
print(x)
# "axb" "b" "c" "dx"
```

`%na<-%` *Assign a value to a vector's missing values*

Description

`%na<-%` is a shortcut to assign a value to all `NA` elements of `x`. The replacement may be a single value (recycled to every `NA`), a vector with one entry per missing element (filled in order), or a vector the same length as `x` (its values at the missing positions are used).

Usage

```
x %na<-% value
```

Arguments

`x`	a vector
`value`	a single value, a vector with one entry per `NA`, or a vector the same length as `x`

Value

Used for the side effect of reassigning `x` in the calling environment; returns the modified `x` invisibly.

Author(s)

Ben Wiseman, <benjamin.h.wiseman@gmail.com>

Examples

```
x <- c("a", NA, "c")
x %na<-% "b"
print(x)
# "a" "b" "c"

x <- c(1, NA, 3, NA)
x %na<-% c(2, 4)   # one replacement per NA, in order
print(x)
# 1 2 3 4
```

Index