Package ‘roperators’

July 20, 2023

Title Additional Operators to Help you Write Cleaner R Code

Version 1.3.14

Maintainer Ben Wiseman <benjamin.wiseman@kornferry.com>

Description Provides string arithmetic, reassignment operators, logical operators that handle missing values, and extra logical operators such as floating point equality and all or nothing. The intent is to allow R users to write code that is easier to read, write, and maintain while providing a friendlier experience to new R users from other language backgrounds (such as 'Python') who are used to concepts such as x += 1 and 'foo' + 'bar'.
Includes operators for not in, easy floating point comparisons, === equivalent, and SQL-like like operations (), etc.
We also added in some extra helper functions, such as OS checks, pasting in Oxford comma format, and functions to get the first, last, nth, or most common element of a vector or word in a string.

License MIT + file LICENSE

Copyright Korn Ferry International

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

Depends R (>= 3.0.0)

Imports stats, tools

Suggests magrittr, knitr, markdown, rmarkdown, prettydoc, rvest

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.1

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' 'utils.R'

NeedsCompilation no

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

Repository CRAN

Date/Publication 2023-07-20 05:30:02 UTC

R topics documented:

---

assign_ops *Assignment operators*

Description

Modifies the stored value of the left-hand-side object by the right-hand-side object. Equivalent of operators such as `+=` `-=` `*=` `/=` in languages like c++ or python. `%+=%` and `%-=%` can also work with strings.

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 stored value by

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.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 doccumentation!

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

   x <- "ab"

   x %+=% "c"

   x %-=% "b"

   x == "ac" # TRUE

# %-=% can also take regular expressions

   x <- "foobar"
```

```
x %-=% "[f|b]"

print(x)
# "ooar"
```

choose_permute *Choose and permute*

Description

Shorthand for some common mathematical operators

Usage

```
n %C% k

n %P% k
```

Arguments

n	whole number (from n choose/permute k)
k	whole number (from n choose/permute k)

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.com>

Examples

```
# Calculate 5 choose 3
print(5 %C% 3)
# Calculate 5 permute 3
print(5 %P% 3)
```

chr *Cleaner conversion functions*

Description

Cleaner conversion functions

convert x to arbitrary class

Usage

```
chr(x, ...)

int(x, ...)

dbl(x, ...)

num(x, ...)

bool(x, ...)

as.class(x, class)
```

Arguments

`x`	object to be converted
`...`	other args for as. conversion
`class`	chatracter name of the class to convert x to

Note

These are shorthand aliases for common conversions There is nothing magical here, but it can make your code more readable

Author(s)

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

Ben Wiseman, <benjamin.wiseman@kornferry.com>

Examples

```
chr(42)   # "42" = as.character
int(42.1) # 42L  = as.integer
dbl("42L") # 42.0 = as.double
num("42") # 42   = as.numeric
bool(42)  # TRUE = as.logical


foo <- 255
as.class(foo, "roman")
# [1] CCLV
```

`comparisons` *Enhanced comparisons*

Description

These operators introduce improved NA handling, reliable floating point tests, and intervals. Specifically:

Equality that handles missing values

Floating point equality, an important bit of functionality missing in base R (`%~=%`)

Strict (value and type) equality, for those familiar with Javascript ===

Greater/less than or equal to with missing value equality

Greater/less than or equal to with floating point and missing equality

Between (ends excluded)

Between (ends included)

Usage

```
x %==% y

x %===% y

x %>=% y

x %<=% y

x %><% y

x %>=<% y
```

Arguments

x	a vector
y	a vector

Author(s)

Ben Wiseman, `<benjamin.wiseman@kornferry.com>`

See Also

Other comparisons: `floating_point_comparisons`

Examples

```
 ## Greater/Less than | Equal

 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's ===
 # Only tru 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


 # NOTE parentheses surrounding expression before this operator are necessary
 # Without parentheses it would be interpreted as .1 + .1 + (.1 %~=% .3)


 #### 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

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
```

`content_checks` *Contents of Vector Checks*

Description

Misc/useful functions to easily determine what is contained in a vector.

Usage

```
is.constant(x)

is.binary(x)
```

Arguments

x object to be tested

Value

a logical value

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

Description

Convert factor with numeric labels into numeric vector

Usage

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

Arguments

x a factor with numeric labels

Author(s)

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

Ben Wiseman, <benjamin.wiseman@kornferry.com>

Examples

```
 x <- factor(c(11, 22, 33, 99))
 as.numeric(x)
 # 1 2 3 4   # NOT typically the desired.expected output

 f.as.numeric(x)
 # 11 22 33 99  # Typically desired output

 # Or...
 as.numeric(as.character(x)) # A tad unsightly
```

`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

These are an important set of operators missing from base R. In particular, using == on two noninterger numbers can give unexpected results (see examples.)

See this for details: https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Usage

```
x %~=% y

x %>~% y

x %<~% y
```

Arguments

x	numeric
y	numeric

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.com>

See Also

Other comparisons: comparisons

Examples

```
## Floating point test of equality ####

# Basic Equality - no roperators:
(0.1 + 0.1 + 0.1) == 0.3   # FALSE
# Basic Equality - with roperators:
(0.1 + 0.1 + 0.1) %~=% 0.3 # TRUE


# NOTE: for floating point >= and <=
(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 approx equal
```

```
(0.1 + 0.1 + 0.1) %>~% 0.3 # TRUE
(0.1 + 0.1 + 0.1) %<~% 0.3 # TRUE
```

get_1st *Little functions to replace common minor functions. useful in apply sttements*

Description

Little functions to replace common minor functions. useful in apply sttements

Get most common thing(s)

Return number of unique things in x

Return vector of n points evenly spaced around the origin point

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 = ""
)

n_unique(x, na.rm = FALSE)

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

Arguments

`x` vector

`type` 'v' (default) for vector `x[1]`; 'l' for list `x[[1]]`

`n` number of points to create

`split` character that separated words. Default = ' '

`collapse` OPTIONAL character - paste output into single string with collapse

`ignore.punct` logical - ignore punctuation marks

`ignore.case` logical - ignore case (if true, will return in lower)

`punct.regex` character - regex used to remove punctuation (by default `[[:punct:]]`)

`punct.replace` character - what to replace punctuation with (default is "")

`na.rm` whether to ignore NAs when determining uniqueness

`origin` number to center sequence around

`spacing` distance between any two points in the sequence

Value

a vector of most common element(s). Will be character unless x is numeric and you don't tell it to collapse into a single string!

a vector of most common element(s). Will be character unless x is numeric and you don't tell it to collapse into a single string!

Numeric vector. Will default to 1 if arguments are left blank to conform with default seq() behaviour.

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.com>

Examples

```
# listr of 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 if you just want to 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"


my_stuff <- c(1:10, 10, 5)
# These are straight forward
get_1st(my_stuff)
get_nth(my_stuff, 3)
get_last(my_stuff)
get_most_frequent(my_stuff)
my_chars <- c("a", "b", "b", "a", "g", "o", "l", "d")
get_most_frequent(my_chars)
get_most_frequent(my_chars, collapse = " & ")
generic_string <- "Who's A good boy? Winston's a good boy!"

get_1st_word(generic_string)
get_nth_word(generic_string, 3)
get_last_word(generic_string)
# default ignores case and punctuation
get_most_frequent_word(generic_string)
# can change like so:
get_most_frequent_word(generic_string, ignore.case = FALSE, ignore.punct = FALSE)
```

integrate *Inline integration*

Description

inline call to integrate that returns integration value rather than list

Usage

```
f %integrate% range
```

Arguments

f	function (with numeric return)
range	vector of two numbers c(low, high)

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.com>

Examples

```
f <- function(x) x^2
print(f %integrate% c(0, 1))
# vs base
x <- integrate(f, 0, 1)
str(x)
```

`library.force` *loads package if available, else tries to install it (from CRAN by default)*

Description

loads package if available, else tries to install it (from CRAN by default)

loads package if available, else tries to install it (from CRAN by default)

Usage

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

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

Arguments

`pkg`	name of package to load/install
`...`	other args used by install.packages

`logicals` *Logical operators*

Description

These are some convienience functions, such as a not-in, and xor operator.

This takes two arguments just like `grepl` - a string and a pattern. TRUE if `grepl(pattern, x, ignore.case=TRUE)` would be TRUE

This takes two arguments just like `grepl` - a string and a pattern. TRUE if `grepl(pattern, x, ignore.case=FALSE, perl=TRUE)` would be TRUE. It's like `%like%` from data.table (but slower, preferably use data.table).

Usage

```
x %ni% y

x %xor% y

x %aon% y

x %rlike% pattern

x %perl% pattern
```

Arguments

`x`	a character vector
`y`	a vector
`pattern`	a single character expression

Note

data.table has a %like% operator which you should try to use instead if working with data.table!

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.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

# Apply a regular expression/substitution to x:

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

 # where x has an O

 x[x %rlike% "O"]

 # [1] "foo" "dOe" "obo"
 # find x where middle letter is "O"

 x[x %rlike% "[a-z]O[a-z]"]

 # will print [1] "foo" "dOe"

# Apply a regular expression/substitution to x:

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

 # find x where middle letter is upper-case "O"

 x[x %perl% "[a-z]O[a-z]"]

 # will print [1] "dOe"
```

os *Operating system checks*

Description

Determine the current operating system as well as provide flags to indicate whether the operating system is a Mac/Windows/Linux.

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 - how do you want to display the age? e.g. years or months?
`rounding`	integer - how many decimal points do you want to see. e.g. 0.25 years

Author(s)

Ben Wiseman, <`benjamin.wiseman@kornferry.com`>

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

Examples

```
# determine operating system
get_os()

# do we have a particular operating system
is.os_mac()
is.os_win()
is.os_lnx()
is.os_unx()
```

`paste_and_cat` *New Paste and Cat Rules*

Description

The available functions are:

`paste_()`is the same as `paste0` but uses an underscore to separate

`cat0()` is analogous to `paste0` but for cat

`catN()` is the same as `cat0` but automatically inserts a new line after the cat

`paste_series()` paste a series of things with a conjunction

`paste_oxford()` shortcut for `paste_series` as 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_`.

`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` indicates the ending conjunction. e.g. setting to "and" would make c("a", "b", "c") paste into "a, b, and c"

`use_oxford_comma` logical - do you want to use an oxford comma at the end?

Author(s)

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

Examples

```
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)
```

`read.tsv` *like read.csv, but for tsv and default header = TRUE*

Description

like read.csv, but for tsv and default header = TRUE

like read.csv, but for pipe-delineated and defaults to header = TRUE

Usage

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

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

Arguments

`file`	path of file you want to load
`...`	other args used by read.table

`string_arithmetic` *String operators*

Description

Perform string concatenation and arithmetic is a similar way to other languages. String division is not present in languages like Python, although arguably it is more useful than string multiplication and can be used with regulr expressions.

Usage

```
x %+% y

x %-% y

x %s*% y

x %s/% y
```

Arguments

x	a string
y	a string

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.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

Misc/useful type checks to prevent duplicated code

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)

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

```
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
na.rm	If true, NA values aren't considered bad for calculations
...	Values to be testes
y	object to be tested

Value

a logical value

Author(s)

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

%regex<-% *Assign to vector only where regular expression is matched*

Description

This takes two arguments just like gsub - a patterns and a replacement. It will totally overwrite any element where the pattern is matched with the second. If you want to simply apply a regex (i.e. replace only the specific bit that matches), use %regex=% instead. If you want to replace with nothing (""), just just %-% or %-=% instead.

Usage

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

Arguments

`x`	a character vector
`value`	c(pattern, replacement)

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.com>

Examples

```
# Overwrite elements that match regex:

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

 # overwrite any element containing a number

 x %regex<-% c("\\d+", "x")

 print(x)

 # "x" "b" "c" "x"
```

`%regex=%` *Modify existing object by regular expression*

Description

This takes two arguments just like `gsub` - a patterns and a replacement. It will only overwrite the parts of any character where the pattern is matched with the second argument. If you want to overwrite whole elements via a regex (i.e. replace the entire element if it matches), use `%regex<-%` instead.

Usage

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

Arguments

`x`	a character vector
`value`	c(pattern, replacement)

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.com>

Examples

```
# Apply a regular expression/substitution to x:

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

 # change any number to "x"

  x %regex=% c("\\d+", "x")

 print(x)

 # "axb" "b" "c" "dx"
```

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

Description

`%na<-%` is a simple shortcut to assign a specific value to all NA elements contained in x.

Usage

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

Arguments

`x`	a vector
`value`	value to replace vector's missing values with

Author(s)

Ben Wiseman, <benjamin.wiseman@kornferry.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)

 print(x)
 # 1 2 3 4
```

Index