

# Package ‘roperators’

May 9, 2026

**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] (ORCID:  
<<https://orcid.org/0000-0002-2908-1188>>),  
Jeff Jones [aut, led]

**Repository** CRAN

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

## Contents

assign_ops . . . . .	2
choose_permute . . . . .	4
chr . . . . .	4
comparisons . . . . .	6
complete_cases . . . . .	7
content_checks . . . . .	9
f.as.numeric . . . . .	10
file_checks . . . . .	11
floating_point_comparisons . . . . .	12
get_1st . . . . .	13
integrate . . . . .	15
library.force . . . . .	16
logicals . . . . .	16
os . . . . .	18
paste_and_cat . . . . .	19
read.tsv . . . . .	21
string_arithmetic . . . . .	21
type_checks . . . . .	22
%regex<-% . . . . .	23
%regex=% . . . . .	24
%na<-% . . . . .	25

**Index** 26

---

assign_ops	<i>Assignment operators</i>
------------	-----------------------------

---

### 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 documentation!

# %+=% 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)  
# "oar"
```

---

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	character 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](mailto: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 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

# 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

```

**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 <a href="#">date</a> , <a href="#">date-time</a> and <a href="#">time interval</a> objects. Complex vectors are allowed for <code>trim = 0</code> , 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 <code>y = x</code> (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	<i>File Extension Checks</i>
-------------	------------------------------

---

**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 non-interger numbers can give unexpected results (see examples.)

See this for details: [https://docs.oracle.com/cd/E19957-01/806-3568/ncg\\_goldberg.html](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	<i>Little functions to replace common minor functions. useful in apply sttements</i>
---------	--

---

### 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(row.names(mtcars)[1:5])
#[1] "Mazda" "Mazda" "Datsun" "Hornet" "Hornet"

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

get_nth_word(row.names(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)

```

---

<code>library.force</code>	<i>loads package if available, else tries to install it (from CRAN by default)</i>
----------------------------	--

---

**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**

<code>pkg</code>	name of package to load/install
<code>...</code>	other args used by <code>install.packages</code>

---

<code>logicals</code>	<i>Logical operators</i>
-----------------------	--------------------------

---

**Description**

These are some convenience 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", "d0e", "rei", "mei", "obo")

# where x has an 0

x[x %rlike% "0"]

# [1] "foo" "d0e" "obo"
```

```
# find x where middle letter is "0"
x[x %rlike% "[a-z]0[a-z]"]

# will print [1] "foo" "d0e"

# Apply a regular expression/substitution to x:
x <- c("foo", "bar", "d0e", "rei", "mei", "obo")

# find x where middle letter is upper-case "O"
x[x %perl% "[a-z]O[a-z]"]

# will print [1] "d0e"
```

---

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 <a href="#">NA_character_</a> .
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	<i>like read.csv, but for tsv and default header = TRUE</i>
----------	---

---

### 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	<i>String operators</i>
-------------------	-------------------------

---

### 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

<code>x</code>	object to be tested
<code>na.rm</code>	If true, NA values aren't considered bad for calculations
<code>...</code>	Values to be testes
<code>y</code>	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"
```

---

<code>%na&lt;-%</code>	<i>Assign value to a vector's missing values</i>
------------------------	--

---

**Description**

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

**Usage**

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

**Arguments**

<code>x</code>	a vector
<code>value</code>	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

## \* comparisons

- comparisons, 6
- floating\_point\_comparisons, 12
- %\*=% (assign\_ops), 2
- %+=% (assign\_ops), 2
- %+% (string\_arithmetic), 21
- %-=% (assign\_ops), 2
- %-% (string\_arithmetic), 21
- %/= % (assign\_ops), 2
- %<=% (comparisons), 6
- %<~% (floating\_point\_comparisons), 12
- %===% (comparisons), 6
- %==% (comparisons), 6
- %><% (comparisons), 6
- %>=% (comparisons), 6
- %>~% (floating\_point\_comparisons), 12
- %C% (choose\_permute), 4
- %P% (choose\_permute), 4
- %^=% (assign\_ops), 2
- %~=% (floating\_point\_comparisons), 12
- %aon% (logicals), 16
- %integrate% (integrate), 15
- %log=% (assign\_ops), 2
- %ni% (logicals), 16
- %perl% (logicals), 16
- %rlike% (logicals), 16
- %root=% (assign\_ops), 2
- %s\*% (string\_arithmetic), 21
- %s/% (string\_arithmetic), 21
- %xor% (logicals), 16
- %na<-%, 25
- %regex<-%, 23
- %regex=%, 24
- all\_cc (complete\_cases), 7
- all\_good\_for\_calcs (type\_checks), 22
- any\_bad\_for\_calcs (type\_checks), 22
- any\_cc (complete\_cases), 7
- as.class (chr), 4

- assign\_ops, 2
- bool (chr), 4
- cat0 (paste\_and\_cat), 19
- catN (paste\_and\_cat), 19
- check\_ext\_against (file\_checks), 11
- choose\_permute, 4
- chr, 4
- colMeans\_cc (complete\_cases), 7
- colSums\_cc (complete\_cases), 7
- comparisons, 6, 12
- complete\_cases, 7
- content\_checks, 9
- cor\_cc (complete\_cases), 7
- cov\_cc (complete\_cases), 7
- date, 9
- date-time, 9
- dbl (chr), 4
- f.as.numeric, 10
- file\_checks, 11
- floating\_point\_comparisons, 6, 12
- get\_1st, 13
- get\_1st\_word (get\_1st), 13
- get\_last (get\_1st), 13
- get\_last\_word (get\_1st), 13
- get\_latest\_CRAN\_version (os), 18
- get\_most\_frequent (get\_1st), 13
- get\_most\_frequent\_word (get\_1st), 13
- get\_nth (get\_1st), 13
- get\_nth\_word (get\_1st), 13
- get\_os (os), 18
- get\_R\_version (os), 18
- get\_R\_version\_age (os), 18
- get\_system\_python (os), 18
- int (chr), 4
- integrate, 15

- IQR\_cc (complete\_cases), 7
- is.atomic\_nan (type\_checks), 22
- is.bad\_and\_equal (type\_checks), 22
- is.bad\_for\_calcs (type\_checks), 22
- is.bad\_for\_indexing (type\_checks), 22
- is.binary (content\_checks), 9
- is.character\_or\_null (type\_checks), 22
- is.constant (content\_checks), 9
- is.df\_or\_null (type\_checks), 22
- is.good\_for\_calcs (type\_checks), 22
- is.good\_for\_indexing (type\_checks), 22
- is.http\_available (os), 18
- is.irregular\_list (type\_checks), 22
- is.list\_or\_null (type\_checks), 22
- is.logical\_or\_null (type\_checks), 22
- is.null\_or\_na (type\_checks), 22
- is.numeric\_or\_null (type\_checks), 22
- is.os\_arm (os), 18
- is.os\_lnx (os), 18
- is.os\_mac (os), 18
- is.os\_unx (os), 18
- is.os\_win (os), 18
- is.os\_x64 (os), 18
- is.R\_revo (os), 18
- is.R\_x64 (os), 18
- is.RStudio (os), 18
- is.scalar (type\_checks), 22
- is.scalar\_or\_null (type\_checks), 22
- is\_csv\_file (file\_checks), 11
- is\_excel\_file (file\_checks), 11
- is\_r\_file (file\_checks), 11
- is\_rda\_file (file\_checks), 11
- is\_rdata\_file (file\_checks), 11
- is\_rds\_file (file\_checks), 11
- is\_spss\_file (file\_checks), 11
- is\_txt\_file (file\_checks), 11
  
- length\_cc (complete\_cases), 7
- library.force, 16
- logicals, 16
  
- mad\_cc (complete\_cases), 7
- max\_cc (complete\_cases), 7
- mean\_cc (complete\_cases), 7
- median\_cc (complete\_cases), 7
- min\_cc (complete\_cases), 7
  
- n\_unique (get\_1st), 13
- n\_unique\_cc (complete\_cases), 7
  
- NA\_character\_, 20
- num (chr), 4
  
- os, 18
  
- paste\_ (paste\_and\_cat), 19
- paste\_and\_cat, 19
- paste\_oxford (paste\_and\_cat), 19
- paste\_series (paste\_and\_cat), 19
- prod\_cc (complete\_cases), 7
  
- quantile\_cc (complete\_cases), 7
  
- range\_cc (complete\_cases), 7
- read.psv (read.tsv), 21
- read.tsv, 21
- require.force (library.force), 16
- rowMeans\_cc (complete\_cases), 7
- rowSums\_cc (complete\_cases), 7
  
- sd\_cc (complete\_cases), 7
- seq\_around (get\_1st), 13
- string\_arithmetic, 21
- sum\_cc (complete\_cases), 7
  
- time interval, 9
- type\_checks, 22
  
- var\_cc (complete\_cases), 7
  
- weighted.mean\_cc (complete\_cases), 7