Package 'roperators'

September 28, 2018

Type Package

Title Additional Operators to Help you Write Cleaner R Code

Version 1.1.0

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'.

Encoding UTF-8

LazyData true

License MIT + file LICENSE

RoxygenNote 6.1.0

Suggests magrittr, knitr, prettydoc

VignetteBuilder knitr

Collate 'type_checks.R' 'operators.R' 'shorthand.R' 'write_pdf.R'

NeedsCompilation no

Author Ben Wiseman [cre, aut, ccp],
Steven Nydick [aut, ccp],
Jeff Jones [aut, led]

Repository CRAN

Date/Publication 2018-09-28 21:40:06 UTC

R topics documented:

`chr` *Cleaner conversion functions*

Description

Cleaner conversion functions

Usage

```
chr(x, ...)

int(x, ...)

dbl(x, ...)

num(x, ...)

bool(x, ...)
```

Arguments

`x`	Value to be converted
`...`	other args for as. conversion
	# Shorthand aliases for common conversions # Nothing magical here, but it can make your code more readable
	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

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>

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
```

is.scalar *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)

is.bad_for_indexing(x)

is.bad_and_equal(x, y)

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

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)

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"
```

%+=% *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

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"
```

`%==%` *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's `===`
- 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

See Also

Other comparisons: %\textasciitilde=%

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
```

`%\textasciitilde=%` *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

Usage

```
x %~=% y

x %>~% y

x %<~% y
```

Arguments

x	numeric
y	numeric

See Also

Other 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
```

`%ni%` *Logical operators*

Description

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

Usage

```
x %ni% y

x %xor% y

x %aon% y
```

Arguments

x	a vector
y	a vector

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
```

%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)

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

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
```

`%+%` *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

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'
```

Index