Package 'infixit'

May 28, 2025

Title Helpful Additional Infix Functions

Version 0.3.1

Description Infix functions in R are those that comes between its arguments such as %in%, +, and *. These are useful in R programming when manipulating data, performing logical operations, and making new functions. 'infixit' extends the infix functions found in R to simplify frequent tasks, such as finding elements that are NOT in a set, in-line text concatenation, augmented assignment operations, additional logical and control flow operators, and identifying if a number or date lies between two others.

License MIT + file LICENSE

URL https://github.com/prlitics/infixit

BugReports https://github.com/prlitics/infixit/issues

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Peter Licari [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-9701-6006>)

Maintainer Peter Licari <prlicari13@gmail.com>

Repository CRAN

Date/Publication 2025-05-28 09:40:02 UTC

Contents

`.is_allFalse` *Tests if an object is entirely comprised of* `FALSE`*s*

Description

This function tests if a passed object is entirely comprised of `FALSE` values.

Usage

```
.is_allFalse(x)
```

Arguments

`x`	The object to test if is entirely comprised of `FALSE` values

Details

This function is exported in order to provide one of the default tests for the `%||%` function and is not really intended for use outside of that context.

Value

A boolean (`TRUE` or `FALSE`)

Examples

```
{
  .is_allFalse(c(FALSE,FALSE,TRUE)) # Will return FALSE
}
```

.is_allNA *Tests if a vector is entirely comprised of NAs*

Description

This function tests if a passed object is entirely comprised of NA values.

Usage

```
.is_allNA(x)
```

Arguments

x	The object to test if is entirely comprised of NA values

Details

This function is exported in order to provide one of the default tests for the %||% function and is not really intended for use outside of that context.

Value

A boolean (TRUE or FALSE)

Examples

```
{
  .is_allNA(c(NA,NA,"NA")) # Will return FALSE
}
```

.is_length_zero *Tests if an object is of length 0*

Description

This function tests if a passed object is of length 0.

Usage

```
.is_length_zero(x)
```

Arguments

x	The object to test if is length(0)

Details

This function is exported in order to provide one of the default tests for the `%|||%` function and is not really intended for use outside of that context.

Value

A boolean (`TRUE` or `FALSE`)

Examples

```
{
  .is_allFalse(c(FALSE,FALSE,TRUE)) # Will return FALSE
}
```

`extended-null-default` *Expanded default operator*

Description

The `%||%` operator will return a default value, defined by the right-hand object, if the left-hand value resolves as `NULL`. However, there may be times when users want more than just `NULL` values to return the default but, also, values that are `NA`, `FALSE`, and those that are length 0 (such as `character(0)` or `integer(0)`).

Usage

```
lhs %|||% rhs
```

Arguments

`lhs`	The left-hand side, the value(s) to be evaluated as.
`rhs`	The right-hand side, the value(s) to be returned if `lhs` evaluates as one of the covered values.

Details

The expanded default operator covers the following cases:

- `NULL`
- An atomic `FALSE`
- A vector where all values are `FALSE`
- An atomic `NA`
- A vector where all values are `NA`
- An object of length 0.

Users have the ability to add additional tests via `options(infixit.extended_default_tests)`. Users can change the current list—including by adding the name of a testing function (i.e., one that returns a Boolean value) that is currently defined in an environment accessible to the function (e.g., in the global environment).

Value

An atomic value or vector the same length as the left-hand side input.

Examples

```
{
  NULL %|||% 'fizzbuzz' #returns fizzbuzz
  FALSE %|||% 'fizzbuzz' #also returns fizzbuzz
  NA %|||% 'fizzbuzz' #still returns fizzbuzz
  'test' %|||% 'fizzbuzz'#returns 'test'
}
```

`null-default` *Default NULL operator*

Description

This operator is seen in `{rlang}` and has been included in base R since version 4.4.0. If the left-hand side is NULL, it will automatically return the value of the right-hand side. This is useful for programming to ensure a function or process returns a non-null default.

Usage

```
x %||% y
```

Arguments

`x`	The left-hand side, the value(s) to be evaluated as either NULL or not.
`y`	The right-hand side, the value(s) to be returned if `lhs` evaluates to NULL.

Value

An atomic value or vector the same length as the left-hand side input.

Examples

```
{
  NULL %||% 'fizzbuzz' #returns fizzbuzz
  'test' %||% 'fizzbuzz'#returns 'test'
}
```

`%btwn%` *Between Infix Operator*

Description

Currently in R, if you want to test if a value is between two others, you have to set it up in a cumbersome manner: `X > Y & X < Z`. `%btwn%` simplifies the operation into a single call: `X %btwn% c(Y, Z)`.

Usage

```
lhs %btwn% rhs
```

Arguments

`lhs`	The left-hand side, the value(s) to be compared.
`rhs`	The right-hand side, the comparative range. Must be a numeric vector of length 2 with the smaller value prior to the larger value. Identical values can be passed.

Details

By default, `%btwn%` evaluates *inclusively*. That is, if the right-hand side is `c(1, 5)` and the left-hand side is `c(1,5)`, it will evaluate as `TRUE TRUE`. If one wants to adjust this default behavior, they can adjust the "infix.btwn" option to be either *inclusive* for the lower-bound ("["), *exclusive* for the lower- bound ("("), *inclusive* for the upper-bound ("]"), or *exclusive* for the upper-bound (")"). To set an inclusive lower-bound but exclusive upper-bound, for example, you would do as follows: `options(infixit.btwn = c("[", ")"))`. Additional options allow you to set which date formats are automatically parsed when comparing if one date is within another (`infixit.btwn.datetimefmt`), and whether `%btwn%` will ignore `NA` values in the comparison or return them as `FALSE` (`infixit.btwn.ignore_na`)

Value

A Boolean vector the same length as the left-hand side input.

Examples

```
{
  13 %btwn% c(12.5, 15) #returns TRUE
}
```

`%-=%` *Subtraction variable reassignment*

Description

Updates the left-hand, numeric type object by subtracting the right-hand value from it, reassigning the difference to the left-hand object.

Usage

```
lhs %-=% rhs
```

Arguments

`lhs`	An numeric object existing in the global/ parent environment.
`rhs`	A numeric value to subtract from the lhs

Details

Currently in R, if you want to update the value of a numeric object to be the outcome of some arithmetic operation, you have to initialize the object and then reassign it. For example: `apple <- 1` and then `apple <- apple - 1`. This sort of thing is generally referred to as augmented variable assignment. This function allows users to update the value of an object through subtracting the value on the right-hand side.

Value

Returns the arithmetically-updated left-hand object into the environment the operation was performed in.

Examples

```
{

example <- 10
example %-=% 3
example # returns 7

}
```

`%nand%` *NAND infix operator*

Description

This is a logical operator that implements NAND (NOT AND).

Usage

```
lhs %nand% rhs
```

Arguments

`lhs`	The left-hand side(s).
`rhs`	The right-hand side value(s).

Details

The NAND truth table is the inverse of the AND table:

LHS	RHS	Value
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE

Value

An atomic value or vector the same length as the left-hand side input.

Examples

```
{
  TRUE %nand% TRUE # Evaluates to FALSE
  FALSE %nand% TRUE # Evaluates to TRUE
  FALSE %nand%FALSE # Evaluates to TRUE
}
```

`%nin%` *Not-In Infix Operator*

Description

This tests whether the elements on the left-hand side is *not* within the elements on the right-hand side. In effect, it is a cleaner, parsimonious way of articulating `!(lhs %in% rhs)`. See the help for `match` for additional documentation on matching.

Usage

```
lhs %nin% rhs
```

Arguments

`lhs`	The left-hand side, element(s) to be sought in the rhs.
`rhs`	The right-hand side; element(s) to be compared against the lhs for possible membership.

Details

Following the convention of `%in%`, which is actually a call to `match`, `%nin%` is defined as: `match(lhs, rhs, nomatch = 0) == 0`. (In the case of `%in%`, the final comparison is `> 0`; as it is looking for indices of the location of `lhs[i]` within `rhs`, any positive match will be greater than 0 by definition since 'R' is a 1-index language rather than a 0-index language such as, e.g., Python).

Value

Returns a Boolean vector the length of lhs conveying whether each element is **un**represented in the elements of rhs.

Examples

```
{

 "apple" %nin% c("carrot", "kiwi" ,"pear")

}
```

`%+=%` *Addition variable reassignment*

Description

Updates the left-hand, numeric type object by adding the right-hand value to it, reassigning the sum to the left-hand object.

Usage

```
lhs %+=% rhs
```

Arguments

`lhs`	An numeric object existing in the global/ parent environment.
`rhs`	A numeric value to add to the sum

Details

Currently in R, if you want to update the value of a numeric object to be the outcome of some arithmetic operation, you have to initialize the object and then reassign it. For example: `apple <- 1` and then `apple <- apple + 1`. This sort of thing is generally referred to as augmented variable assignment. This function allows users to update the value of an object through adding the value on the right-hand side.

Value

Returns the arithmetically-updated left-hand object into the environment the operation was performed in.

Examples

```
{

example <- 5
example %+=% 8
example # returns 13

}
```

`%+%` *Paste Infix Operator*

Description

Many programming languages utilize + as a means of concatenating strings. In standard R, however, + will return an error when used with strings. `%+%` provides this ability for parsimonious string concatenation.

Usage

```
lhs %+% rhs
```

Arguments

`lhs`	The left-hand side.
`rhs`	The right-hand side.

Details

By default, it uses `paste0` under the hood, but this can be shifted to `paste` by running `options(infixit.paste = "paste0")`. By default (as with `paste`), this will have the seperator be a single space (`" "`) between the pasted objects. This behavior can be changed with the `infixit.paste_sep` option. E.g., `options(infixit.paste_sep = "|")`

Value

A string pasting the rhs to the lhs.

Examples

```
{

b <- "An additional sentence."

"This is a sentence. " %+% b
}
```

`%^=%` *Exponentiation variable reassignment*

Description

Updates the left-hand, numeric type object by raising it to the power of the right-hand value, reassigning the result to the left-hand object.

Usage

```
lhs %^=% rhs
```

Arguments

`lhs`	An numeric object existing in the global/ parent environment.
`rhs`	A numeric value to raise the lhs by

Details

Currently in R, if you want to update the value of a numeric object to be the outcome of some arithmetic operation, you have to initialize the object and then reassign it. For example: `apple <- 2` and then `apple <- apple ^ 3`. This sort of thing is generally referred to as augmented variable assignment. This function allows users to update the value of an object through raising it to the power of the value on the right-hand side.

Value

Returns the arithmetically-updated left-hand object into the environment the operation was performed in.

Examples

```
{

example <- 2
example %^=% 3
example # returns 8

}
```

`%/=%` *Division variable reassignment*

Description

Updates the left-hand, numeric type object by dividing it by the right-hand value, reassigning the quotient to the left-hand object.

Usage

```
lhs %/=% rhs
```

Arguments

`lhs`	An numeric object existing in the global/ parent environment.
`rhs`	A numeric value to divide the lhs by

Details

Currently in R, if you want to update the value of a numeric object to be the outcome of some arithmetic operation, you have to initialize the object and then reassign it. For example: `apple <- 10` and then `apple <- apple / 2`. This sort of thing is generally referred to as augmented variable assignment. This function allows users to update the value of an object through dividing the value on the right-hand side.

Value

Returns the arithmetically-updated left-hand object into the environment the operation was performed in.

Examples

```
{

example <- 10
example %/=% 2
example # returns 5

}
```

`%*=%` *Multiplication variable reassignment*

Description

Updates the left-hand, numeric type object by multiplying it by the right-hand value, reassigning the product to the left-hand object.

Usage

```
lhs %*=% rhs
```

Arguments

`lhs`	An numeric object existing in the global/ parent environment.
`rhs`	A numeric value to multiply the lhs by

Details

Currently in R, if you want to update the value of a numeric object to be the outcome of some arithmetic operation, you have to initialize the object and then reassign it. For example: `apple <- 2` and then `apple <- apple * 3`. This sort of thing is generally referred to as augmented variable assignment. This function allows users to update the value of an object through multiplying it by the value on the right-hand side.

Value

Returns the arithmetically-updated left-hand object into the environment the operation was performed in.

Examples

```
{

example <- 3
example %*=% 4
example # returns 12

}
```

`%xor%` *XOR infix operator*

Description

This is a logical operator that implements XOR. (Exclusive or).

Usage

```
lhs %xor% rhs
```

Arguments

`lhs`	The left-hand side(s).
`rhs`	The right-hand side value(s).

Details

The XOR truth-table is as follows:

LHS	RHS	Value
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

In contrast with the standard OR, XOR evaluates to FAlSE if both arguments are TRUE.

Value

An atomic value or vector the same length as the left-hand side input.

Examples

```
{
  TRUE %xor% TRUE # Evaluates to FALSE
  FALSE %xor% TRUE # Evaluates to TRUE
}
```

Index