Package 'infix'

December 25, 2018

Type Package

Title Basic Infix Binary Operators

Version 0.1.0

Description Contains a number of infix binary operators that may be useful in day to day practices.

URL http://github.com/ebeneditos/infix

BugReports http://github.com/ebeneditos/infix/issues

Imports magrittr

Suggests covr, testthat

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

NeedsCompilation no

Author Ernest Benedito [aut, cre]

Maintainer Ernest Benedito <ebeneditos@gmail.com>

Repository CRAN

Date/Publication 2018-12-25 22:20:27 UTC

R topics documented:

except *Simple Error Handling*

Description

Use this method to handle errors. The function evaluates an expression, and if it raises an error then evaluates a second expression.

Usage

```
tryExcept(expr, except = { }, error = function(e) { })

expr %except% except
```

Arguments

expr	Expression to be evaluated.
except	Expression to be evaluated if expr raises an error. By default it is an empty expression.
error	Handler function for an error condition occurred during the evaluation of expr. It's output is not used, as the output in case of an error is the evaluation of except. By default it is an empty function.

Details

tryExcept is a wrapper around tryCatch, but it allows you to evaluate an expression except when an error occurs to the first expression argument expr. Note that, if expr raises an error, the code evaluated before the error will be in use.

Examples

```
# No errors are raised
tryExcept(stop())

# If 'expr' has no errors
tryExcept({
  foo <- "foo"
}, except = {
  foo <- "foo bar"
})
print(foo) # "foo"

# If 'expr' has an error
tryExcept({
  foo <- "foo"
  stop()
}, except = {
  foo <- "foo bar"
```

```
})
print(foo) # "foo bar"

# Running it with the infix operator
{foo <- "foo"} %except% {foo <- "foo bar"}
print(foo) # "foo"

{ foo <- "foo"
  stop()
} %except% {
  foo <- "foo bar"
}
print(foo) # "foo bar"
```

`file.path` *Construct Path to File*

Description

Analogous to `file.path(x, y)`.

Usage

```
x %//% y
```

Arguments

`x, y`	Character vectors.

Examples

```
"home" %//% "dir" # returns "home/dir"
```

`funlogic` *Function Logical Operators*

Description

Creates a function which returns the corresponding logical operation between what `f1` and `f2` return.

Usage

```
f1 %&% f2

f1 %|% f2

f1 %xor% f2
```

Arguments

f1, f2	Arbitrary predicate functions.

Details

To obtain the logical negation of what a function f returns, use base function Negate.

Examples

```
is.null.na <- is.null %|% is.na
all(is.null.na(NA), is.null.na(NULL)) # returns TRUE
```

---

infix *Basic Infix Binary Operators*

Description

Contains a number of infix binary operators that may be useful in day to day practices.

Operators

except Simple Error Handling.

operators Common Infix Operators.

funlogic Function Logical Operators.

pipes Package's 'magrittr' pipe-operators.

Author(s)

Maintainer: Ernest Benedito <ebeneditos@gmail.com>

See Also

Useful links:

- http://github.com/ebeneditos/infix
- Report bugs at http://github.com/ebeneditos/infix/issues

nil *Default value for* NULL

Description

This infix function makes it easy to replace NULL's with a default value. It's inspired by the way that Ruby's or operation (`||`) works.

Usage

```
a %||% b
```

Arguments

a, b	If a is NULL, will return b; otherwise returns a.

Examples

```
1 %||% 2 # returns 1
NULL %||% 2 # returns 2
```

nomatch *Value (non) Matching*

Description

The logical negation of %in%.

Usage

```
x %!in% table
```

Arguments

x	Vector or NULL: the values to be (not) matched.
table	Vector or NULL: the values to be (not) matched against.

Examples

```
4 %!in% 1:3 # returns TRUE
```

operators *Common Infix Operators*

Description

Common Infix Operators

Infix Operators

`paste0` Concatenate Strings (see `?"%+%"`).

`file.path` Construct Path to File (see `?"%//%"`).

`nomatch` Value (non) Matching (see `?"%!in%"`).

`nil` Default value for `NULL` (see `?"%||%"`).

paste0 *Concatenate Strings*

Description

Analogous to `paste0(x, y)`.

Usage

```
x %+% y
```

Arguments

`x, y`	Objects to be converted to character vectors.

Examples

```
"01" %+% "jan" %+% "1970" # returns "01jan1970"
```

Index