Package 'bazar'

March 16, 2019

Type Package

Title Miscellaneous Basic Functions

Version 1.0.11

Date 2019-03-15

Description A collection of miscellaneous functions for
copying objects to the clipboard ('Copy');
manipulating strings ('concat', 'mgsub', 'trim', 'verlan');
loading or showing packages ('library_with_dep', 'require_with_dep',
'sessionPackages');
creating or testing for named lists ('nlist', 'as.nlist', 'is.nlist'),
formulas ('is.formula'), empty objects ('as.empty', 'is.empty'),
whole numbers ('as.wholenumber', 'is.wholenumber');
testing for equality ('almost.equal', 'almost.zero') and computing
uniqueness ('almost.unique');
getting modified versions of usual functions ('rle2', 'sumNA');
making a pause or a stop ('pause', 'stopif');
converting into a function ('as.fun');
providing a C like ternary operator ('condition %?% true %:% false');
finding packages and functions ('get_all_pkgs', 'get_all_funs');
and others ('erase', '%nin%', 'unwhich', 'top', 'bot', 'normalize').

License GPL-3

LazyData TRUE

Depends R (>= 3.1.3)

Imports kimisc, stats, tools, utils

Suggests knitr, testthat

URL https://github.com/paulponcet/bazar

BugReports https://github.com/paulponcet/bazar/issues

RoxygenNote 6.1.0

NeedsCompilation no

Author Paul Poncet [aut, cre]

Maintainer Paul Poncet <paulponcet@yahoo.fr>

Repository CRAN

Date/Publication 2019-03-15 23:33:24 UTC

R topics documented:

almost.equal *Test (almost) equality of numeric values*

Description

The function `almost.equal` tests if two numeric vectors have equal values up to a tolerance.

Usage

```
almost.equal(x, y, tolerance = sqrt(.Machine$double.eps))
```

Arguments

`x`	numeric vector.
`y`	numeric vector of the same length as `x`.
`tolerance`	numeric. Differences smaller than tolerance are considered as equal. The default value is close to `1.5e-8`.

Value

A logical vector of the same length as `x` and `y`.

Author(s)

Tommy on StackOverflow, see http://stackoverflow.com/a/7667703.

Examples

```
almost.equal(x = 1:3,
             y = 1:3 + c(10^(-6), 10^(-7), 10^(-8)))
```

almost.unique *Almost unique elements*

Description

The function `almost.unique` extracts elements of a vector `x` that are unique up to a tolerance factor.

Usage

```
almost.unique(x, ...)

## Default S3 method:
almost.unique(x, tolerance = sqrt(.Machine$double.eps),
  ...)
```

Arguments

x	numeric. The vector of numeric values at stake.
...	Additional arguments to be passed to the function duplicated, which is used internally by almost.unique.
tolerance	numeric. Relative differences smaller than tolerance are considered as equal. The default value is close to 1.5e-8.

Value

A vector of the same type as x.

See Also

unique, duplicated.

Examples

```
almost.unique(c(1, 1.01), tol = 0.1)
almost.unique(c(1, 1.01), tol = 0.01)

almost.unique(c(1, 2, 3), tol = 10)
almost.unique(c(1, 2, 3), tol = 5)
almost.unique(c(1, 2, 3), tol = 1)
```

---

almost.zero *Test if values of a vector are almost zero*

Description

The function almost.zero tests if values of the numeric vector x are equal to zero up to a tolerance.

Usage

```
almost.zero(x, tolerance = sqrt(.Machine$double.eps))
```

Arguments

x	numeric. The vector of numeric values at stake.
tolerance	numeric. Differences smaller than tolerance are considered as equal. The default value is close to 1.5e-8.

Value

A logical vector of the same length as x.

See Also

all.equal.

Examples

```
almost.zero(c(0, 10^(-7), 10^(-8)))
```

---

as.empty *Convert to an empty object*

Description

Convert x to an empty object.

Usage

```
as.empty(x, ...)

## Default S3 method:
as.empty(x, ...)

## S3 method for class 'data.frame'
as.empty(x, ...)
```

Arguments

x	An object.
...	Additional parameterS.

Value

An empty object

See Also

is.empty in this package.

Examples

```
x <- c("a", "b", "c")
as.empty(x)
class(as.empty(x)) # still a character

x <- factor(LETTERS)
as.empty(x)        # levels are kept
class(as.empty(x)) # still a factor
```

```
x <- data.frame(x = 1:3, y = 2:4)
as.empty(x)
```

`as.fun` *Convert object to function*

Description

`as.fun` is a generic function that does the same as `as.function` from package **base**, with the additional feature that `as.fun.character` converts a string into the function it names.

Usage

```
as.fun(x, ...)

## Default S3 method:
as.fun(x, envir = parent.frame(), ...)

## S3 method for class 'character'
as.fun(x, ...)

## S3 method for class 'name'
as.fun(x, ...)

## S3 method for class 'call'
as.fun(x, ...)

## S3 method for class 'numeric'
as.fun(x, ...)

## S3 method for class 'logical'
as.fun(x, ...)

## S3 method for class 'factor'
as.fun(x, ...)

## S3 method for class 'complex'
as.fun(x, ...)

## S3 method for class 'data.frame'
as.fun(x, ...)

## S3 method for class 'lm'
as.fun(x, ...)

## S3 method for class 'rpart'
as.fun(x, ...)
```

Arguments

x	The object to convert.
...	Additional arguments (currently not used).
envir	Environment in which the function should be defined.

Value

The desired function.

Author(s)

`as.fun.character` is adapted from MrFlick, see https://stackoverflow.com/a/38984214 on StackOverflow.

Examples

```
as.fun(mean)
as.fun("mean")
as.fun("edit")
as.fun("stats::predict")

## the constant function '1'
f <- as.fun(1)
f(2)   # 1
f("a") # 1

## the constant function 'FALSE'
f <- as.fun(FALSE)
f(2)   # FALSE
f("a") # FALSE

f <- as.fun(data.frame(x = 1:2, y = 2:3))
f("x") # 'x' column
f("y") # 'y' column
```

---

as.na *Transform values to NA*

Description

These methods transform values to NA for different classes of objects.

Usage

```
as.na(x, ...)

## Default S3 method:
as.na(x, ...)

## S3 method for class 'data.frame'
as.na(x, ...)

## S3 method for class 'list'
as.na(x, ...)
```

Arguments

x	The object at stake.
...	Additional arguments (unused).

Value

An object of the same class as x; the attributes of x are passed unchanged to the result.

Examples

```
x <- c("a", "b", "c")
as.na(x)
class(as.na(x)) # still a character

x <- factor(LETTERS)
as.na(x)        # levels are kept
class(as.na(x)) # still a factor

x <- data.frame(x = 1:3, y = 2:4)
as.na(x)
dim(as.na(x))

x <- matrix(1:6, 2, 3)
attr(x, "today") <- Sys.Date()
as.na(x)        # attributes are kept
```

---

bazar *bazar: miscellaneous basic functions*

Description

bazar provides a collection of miscellaneous functions for

- copying objects to the clipboard (Copy);

- manipulating strings (concat, mgsub, trim, verlan);
- loading or showing packages (library_with_dep, require_with_dep, sessionPackages);
- creating or testing for named lists (nlist, as.nlist, is.nlist), formulas (is.formula), empty objects (as.empty, is.empty), whole numbers (as.wholenumber, is.wholenumber);
- testing for equality (almost.equal, almost.zero);
- getting modified versions of usual functions (rle2, sumNA);
- making a pause or a stop (pause, stopif);
- and others (erase, %in%, unwhich).

`concat` *String concatenation*

Description

The function `concat` concatenates character vectors all together.

`concat0(.)` is a wrapper for `concat(., sep = "")`. `concat_(.)` is a wrapper for `concat(., sep = "_")`.

Usage

```
concat(..., sep = " ", na.rm = TRUE)

concat0(..., na.rm = TRUE)

concat_(..., na.rm = TRUE)
```

Arguments

`...`	One or more objects, to be converted to character vectors and concatenated.
`sep`	character. The character to use to separate the result.
`na.rm`	logical. If `TRUE` (the default), missing values are removed before concatenation.

Value

Always a character value (vector of length `1`).

See Also

paste.

Examples

```
v <- c("Florence", "Julie", "Angela")
concat0(v)
concat_(v)
concat(v, sep = "^^")
concat0(c("a", "b"), c(1, NA, 3), NA)
concat(c(NA, NA))
concat(c(NA, NA), na.rm = FALSE) # usually not desirable
```

Copy *Copy data to the clipboard*

Description

The function Copy can typically be used to copy data from a data frame, in order to paste it somewhere else (in Excel for instance).

Usage

```
Copy(x, size = 128L, quote = TRUE, sep = "\t", na = "",
  dec = ".", ...)
```

Arguments

x	An object.
size	integer. Number of kilobytes. Increase this value if the object x is too big.
quote	See the eponymous argument in write.table.
sep	character. The field separator string.
na	character. The string to use for missing values.
dec	character. The string to use for decimal points in numeric or complex columns.
...	Additional arguments to be passed to write.table.

---

erase *Delete objects*

Description

The function erase deletes all objects that live in the calling environment.

Usage

```
erase(ask = TRUE)
```

Arguments

ask	logical. If TRUE (the default), a confirmation is interactively asked to the user.

Warning

use this function with care!

---

get_all_funs *Functions exported by a package*

Description

get_all_funs provides all the functions exported by a given installed package.

Usage

```
get_all_funs(pkg)
```

Arguments

pkg	character. The package of interest. (Must be installed already.)

Value

A character vector, the functions exported.

Examples

```
get_all_funs("stats")
```

---

get_all_pkgs *Packages exporting a function*

Description

get_all_pkgs provides all packages (belonging to a given list of packages) exported by a given function.

Usage

```
get_all_pkgs(fun, packages = NULL)
```

Arguments

fun	function or character. The function of interest.
packages	The packages to look into. If NULL, the list of currently attached packages is explored.

Value

A character vector, the packages.

Examples

```
## Not run:
get_all_pkgs("as.fun")
get_all_pkgs(as.fun)
get_all_pkgs("stats::median")

## End(Not run)
```

---

get_vars *Get formula variables*

Description

The function get_vars extracts variable names from a formula.

Usage

```
get_vars(formula, data = NULL, intersection = TRUE)
```

Arguments

formula	a formula.
data	data.frame or matrix. If not NULL, formulas with a dot . are permitted.
intersection	logical. If TRUE and data is not NULL, the intersection between variables found in the formula and data column names is returned.

Value

a character vector, the variables found.

See Also

all.vars, get.vars

Examples

```
get_vars(y ~ x1 + x2 - x1)
get_vars(y ~ . - x1, data = data.frame(y = 1, x1 = 2, x2 = 3))
get_vars(y + z ~ x1 + x2 - x1 | x3)
get_vars(y ~ x1 + I(log(x2)))
get_vars(y ~ x1*x2)
get_vars(y ~ x1:x2)
get_vars(~ x1 + x2)
```

`is.empty` *Test emptyness*

Description

These methods test if an object `x` is empty.

Usage

```
is.empty(x)

## Default S3 method:
is.empty(x)

## S3 method for class 'data.frame'
is.empty(x)
```

Arguments

`x`	An object to be tested.

Value

`TRUE` if `x` is empty, `FALSE` otherwise.

See Also

`as.empty` in this package.

Examples

```
is.empty(4)
is.empty(c())
is.empty(new.env())
is.empty(character(0))
is.empty(list())
is.empty(integer(0))
is.empty(data.frame())
```

is.formula *Test if an object is a formula*

Description

The function `is.formula` tests if the object `x` is a formula.

Usage

```
is.formula(x)
```

Arguments

`x`	An object.

Value

A logical, `TRUE` if `x` is a formula.

Examples

```
is.formula("this is a formula")
is.formula(f <- formula("y ~ x"))
is.formula(update(f, ~ . -1))
```

is.wholenumber *Test if the values of a vector are whole numbers*

Description

The function `is.wholenumber` tests if values of the numeric vector `x` are all whole numbers (up to a tolerance).

The function `as.wholenumber` is a synonym for `as.integer`.

Usage

```
is.wholenumber(x, tolerance = sqrt(.Machine$double.eps))

as.wholenumber(x, ...)
```

Arguments

`x`	a vector to be tested.
`tolerance`	numeric. Differences smaller than tolerance are considered as equal. The default value is close to `1.5e-8`.
`...`	Additional arguments passed to or from other methods.

Value

A logical, TRUE if all values of x are (finite) whole numbers. If x contains NA or NaN, then NA is returned.

Examples

```
x = c(1L, 10L)
is.integer(x)
is.wholenumber(x)

x = c(1, 10)
is.integer(x)
is.wholenumber(x) # here is the difference with 'is.integer'

is.wholenumber(1+10^(-7))
is.wholenumber(1+10^(-8))
```

---

isNA *Test if NA*

Description

isNA tests if an object x is identical to one of NA, NA_character_, NA_complex_, NA_integer_, NA_real_, or NaN.

Usage

```
isNA(x)
```

Arguments

x	An R object.

Value

TRUE or FALSE.

See Also

isTRUE.

library_with_dep *Loading/Attaching and listing of packages with dependencies*

Description

library_with_dep and require_with_dep behave respectively like library and require, but also load and attach dependent packages (typically packages listed in the Imports field of the DESCRIPTION file).

Usage

```
library_with_dep(package, help, pos = 2, lib.loc = NULL,
  character.only = FALSE, logical.return = FALSE,
  warn.conflicts = TRUE, quietly = FALSE,
  verbose = getOption("verbose"), which = "Imports",
  recursive = FALSE, reverse = FALSE)

require_with_dep(package, lib.loc = NULL, quietly = FALSE,
  warn.conflicts = TRUE, character.only = FALSE, which = "Imports",
  recursive = FALSE, reverse = FALSE, verbose = getOption("verbose"))
```

Arguments

package	the name of a package, given as a name or literal character string, or a character string, depending on whether character.only is FALSE (default) or TRUE.
help	the name of a package, given as a name or literal character string, or a character string, depending on whether character.only is FALSE (default) or TRUE.
pos	the position on the search list at which to attach the loaded namespace. Can also be the name of a position on the current search list as given by search().
lib.loc	character. A vector describing the location of R library trees to search through, or NULL. The default value of NULL corresponds to all libraries currently known to .libPaths(). Non-existent library trees are silently ignored.
character.only	logical. Indicates whether package or help can be assumed to be character strings.
logical.return	logical. If it is TRUE, then FALSE or TRUE is returned to indicate success.
warn.conflicts	logical. If TRUE, warnings are printed about conflicts from attaching the new package. A conflict is a function masking a function, or a non-function masking a non-function.
quietly	logical. If TRUE, no message confirming package attaching is printed, and most often, no errors/warnings are printed if package attaching fails.
verbose	logical. If TRUE, additional diagnostics are printed.
which	character. A vector listing the types of dependencies, a subset of c("Depends", "Imports", "LinkingTo Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances".

recursive	logical. Should (reverse) dependencies of (reverse) dependencies (and so on) be included?
reverse	logical. If FALSE (default), regular dependencies are calculated, otherwise reverse dependencies.

See Also

library and require from package **base**; package_dependencies from **tools**; installed.packages from **utils**.

---

mgsub *Multiple gsub*

Description

The function mgsub is a 'multiple' version of gsub.

Usage

```
mgsub(pattern, replacement, x, ...)
```

Arguments

pattern	character vector containing regular expressions to be matched in the given character vector.
replacement	a replacement vector of the same length as pattern for matched pattern. Coerced to character if possible.
x	vector or NULL: the values to be matched against.
...	additional parameters to be passed to gsub.

Value

A character vector of the same length as x.

Author(s)

Theodore Lytras on StackOverflow, see http://stackoverflow.com/a/15254254/3902976

See Also

gsub from package **base**.

Examples

```
mgsub(c("aa", "AA"), c("bb", "BB"), c("XXaaccAACC", "YYaaccAACC", "ZZaaccAACC"))
```

nlist *Named lists*

Description

Functions to construct, coerce and check for named lists.

Usage

```
nlist(...)

as.nlist(x, ...)

is.nlist(x)
```

Arguments

...	Named objects.
x	Object to be coerced or tested.

Value

A named list.

Examples

```
x <- nlist(x = 2, y = c("a", "b"))
is.nlist(x)
```

normalize *Normalize a numeric vector*

Description

This function divides x by the result of fun(x).

Usage

```
normalize(x, fun = "max", na.rm = TRUE, ...)
```

Arguments

x	numeric. A vector.
fun	character or function. Should own an na.rm argument. fun(x) should return either one unique value, or a numeric vector of the same length as x.
na.rm	Should missing values be removed in the calculation of fun(x)?
...	Additional arguments to be passed to fun.

Value

A numeric vector of the same length as x.

Examples

```
x <- rnorm(10)
normalize(x)
```

---

pause *Have a rest, make a pause*

Description

The pause function stops momentarily the execution of a program. Pressing <Enter> continues the execution; typing 'stop' (without quotation marks) ends the program.

Usage

```
pause(duration = Inf)
```

Arguments

duration	numeric or infinite. If duration is infinite (the default), then a pause is made until the user presses <Enter> or types 'stop'. Else if x = duration is a number, then a pause is made during x seconds.

See Also

Sys.sleep.

rle2 *Run length encoding (modified version)*

Description

Compute the lengths and values of runs of `almost.equal` values in a vector.

Usage

```
rle2(x, tolerance = sqrt(.Machine$double.eps))
```

Arguments

`x`	numeric vector.
`tolerance`	numeric. Differences smaller than tolerance are considered as equal. The default value is close to `1.5e-8`.

Value

An object of class `"rle"` which is a list with components:

- `lengths`: an integer vector containing the length of each run.
- `values`: a vector of the same length as lengths with the corresponding values.

See Also

`almost.equal` in this package; `rle` in package **base**.

rollfun *Moving windows with custom function*

Description

Windowed / rolling operations on a vector, with a custom function `fun` provided as input.

Usage

```
rollfun(x, k, fun = "mean", ..., .idx = NULL)

make_idx(k, n)
```

Arguments

x	A vector.
k	integer. Width of moving window; must be an integer between one and length(x).
fun	character or function. The function to be applied on moving subvectors of x.
...	Additional arguments to be passed to fun.
.idx	integer. A vector of indices that can be precalculated with the function make_idx.
n	integer. Length of the input vector x.

See Also

Functions roll_mean and others in package **RcppRoll** for a more efficient implementation of rollfun to specific values of fun.

Similarly, see functions rollmean and others in package **zoo** and functions runmean and others in package **caTools**.

Examples

```
set.seed(1)
x <- sample(1:10)
rollfun(x, k = 3)
rollfun(x, k = 3, fun = max)
```

---

sessionPackages *Shows packages attached to the current R session*

Description

The function sessionPackages prints the list of packages attached to the current R session.

Usage

```
sessionPackages(package = NULL)
```

Arguments

package	a character vector naming installed packages, or NULL (the default) meaning all attached packages.

Details

This function reuses part of the code from sessionInfo.

Value

A list with the following components:

- basePkgs: a character vector of base packages which are attached.
- otherPkgs (not always present): a character vector of other attached packages.

See Also

sessionInfo from package **utils**, R.version from package **base**.

Examples

```
sessionPackages()
```

---

stopif *Ensure that R expressions are false*

Description

If any of the expressions in ... are not all FALSE, stop is called, producing an error message indicating the first of the elements of ... which were not false.

Usage

```
stopif(...)
```

Arguments

...	Any number of (logical) R expressions, which should evaluate to TRUE.

Value

(NULL if all statements in ... are FALSE.)

See Also

stopifnot from package **base**.

Examples

```
## Not run:
stopif(is.empty(c(2,1)), 4 < 3) # all FALSE
stopif(is.empty(numeric(0)))

## End(Not run)
```

sumNA *Modified sum of vector elements*

Description

The function sumNA returns the sum of all the values in its arguments. Contrarily to sum, it returns NA instead of 0 when the input contains only missing values and missing values are removed.

Usage

```
sumNA(..., na.rm = FALSE)
```

Arguments

...	numeric or complex or logical vectors.
na.rm	logical. Should missing values (including NaN) be removed?

Value

The sum. Returns NA if x contains only missing values and na.rm = TRUE.

See Also

sum.

Examples

```
x <- c(NA, NA)
sum(x)
sumNA(x)
sum(x, na.rm = TRUE)
sumNA(x, na.rm = TRUE) # here is the difference with 'sum()'

sum(c())
sumNA(c())
```

top *Top or bottom element of an object*

Description

top(x) is an alias for head(x, 1L). bot(x) is an alias for tail(x, 1L).

Usage

```
top(x)

bot(x)
```

Arguments

x	an object.

Value

An object (usually) like x but generally smaller.

See Also

head and head from package **utils**

`trim` *Removes extra whitespaces from a string*

Description

The function `trim` removes unnecessary whitespaces from a character vector.

Usage

```
trim(x)
```

Arguments

x	character. The character vector at stake.

Value

A character vector of the same length as x.

See Also

gsub.

Examples

```
trim(c(" a b", "Hello  World "))
```

unwhich *Quasi-inverse of the 'which' function*

Description

The unwhich function is a kind of inverse of the which function.

Usage

```
unwhich(w, n)
```

Arguments

w	A vector of integers; morally the result of a call to which.
n	integer. The length of the result; morally the length of the x argument of a call to which.

Value

A logical vector of length n.

See Also

which.

Examples

```
x1 <- c(TRUE, FALSE, TRUE, TRUE)
x2 <- unwhich(which(x1), length(x1))
identical(x1, x2) # TRUE

w1 <- c(2, 4, 5, 1, 1)
w2 <- which(unwhich(w1, 10))
identical(sort(unique(as.integer(w1))), w2) # TRUE
```

verlan *Back slang*

Description

The verlan function reverses the order of the characters in a string.

Usage

```
verlan(x)
```

Arguments

x	character. A vector of strings.

Value

A character vector of the same length as x.

Examples

```
verlan("baba") ## "abab"
verlan(c("radar", "paul")) ## c("radar", "luap")
```

%nin% *Value matching*

Description

The function %nin% is the negation of the function %in%.

Usage

```
x %nin% table
```

Arguments

x	vector or NULL: the values to be matched.
table	vector or NULL: the values to be matched against.

Value

A logical vector, indicating if a non-match was located for each element of x: thus the values are TRUE or FALSE and never NA.

See Also

match.

Examples

```
1:10 %nin% c(1,3,5,9)
```

`%?%` *If-Then-Else ternary operator*

Description

This is a C like ternary operator, the syntax being `condition %?% true %:% false`.

Usage

```
condition %?% true

lhs %:% false
```

Arguments

`condition`	logical. A vector.
`true, false`	Values to use for `TRUE` and `FALSE` values of `condition`. They must be either the same length as `condition`, or length `1`.
`lhs`	Left-hand side of `%:%`, which should come from the result of a `%?%` call.

Value

If `length(x) > 1`, then `ifelse` is used.

Author(s)

Richie Cotton, see https://stackoverflow.com/a/8791496/3902976; Paul Poncet for the small modifications introduced.

Examples

```
(capitalize <- sample(c(TRUE, FALSE), 1))
capitalize %?% LETTERS[1:3] %:% letters[1:2]

# Does not work
## Not run:
capitalize %?% 1*1:3 %:% 1:2

## End(Not run)

# Does work
capitalize %?% {1*1:3} %:% 1:2

# Does work too
capitalize %?% (1*1:3) %:% 1:2

# Vectorized version also works
c(capitalize,!capitalize) %?% "A" %:% c("b","c")
```

```
# Chaining operators is permitted
FALSE %?% "a" %:%
  (FALSE %?% "b") %:%
  (capitalize %?% "C") %:% "c"
```

Index