Package 'FastUtils'

July 9, 2024

Type Package

Title Fast, Readable Utility Functions

Version 0.1.1

Date 2024-06-21

Maintainer Qile Yang <qile.yang@berkeley.edu>

Description A wide variety of tools for general data analysis, wrangling, spelling, statistics, visualizations, package development, and more. All functions have vectorized implementations whenever possible.

BugReports https://github.com/Qile0317/FastUtils/issues/

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Imports assertthat, BiocManager, devtools, dplyr, ggplot2, hash, lifecycle, methods, Rcpp (>= 1.0.12), rlang, testthat, tools, usethis

LinkingTo Rcpp

Suggests covr, knitr, rmarkdown, spelling

Config/testthat/edition 3

RoxygenNote 7.3.2

VignetteBuilder knitr

URL https://github.com/Qile0317/FastUtils, https://qile0317.github.io/FastUtils/

NeedsCompilation yes

Author Qile Yang [cre, aut, cph]

Repository CRAN

Date/Publication 2024-07-09 15:40:06 UTC

Contents

add . . . 3
bound . . . 4
closestWord . . . 4
colToRownames . . . 5
createHash . . . 6
createMutator . . . 6
createPkgLoader . . . 7
divide . . . 8
enclose . . . 9
encloseBr . . . 9
enumerateit . . . 10
findMissingRdSections . . . 10
fixColnames . . . 11
getAvgHex . . . 12
getChar . . . 13
getfirst . . . 14
getlast . . . 14
getPkgKeywords . . . 15
getPlotDims . . . 16
getUniquePairsUpTo . . . 16
greplDir . . . 17
ind . . . 18
initDataFrameWithColnames . . . 19
initEmptyTable . . . 19
initList . . . 20
initTestthat . . . 20
initV . . . 21
isBound . . . 22
isCamelCase . . . 22
isEven . . . 23
isOdd . . . 24
isPascalCase . . . 24
isSnakeCase . . . 25
isVowel . . . 25
joinRegex . . . 26
listFiles . . . 27
multiply . . . 27
mutateToRownames . . . 28
namedNumericToTable . . . 29
prependIndefArticle . . . 29
quietly . . . 30
rmByName . . . 31
rownamesToCol . . . 31
scaleHex . . . 32
setColnames . . . 33
setRownames . . . 33

splitCamel 34
splitSnake 35
startsWithVowel 35
stopp 36
stripSpaces 37
substrEnd 37
subtract 38
tableToNumeric 39
test_quietly_that 39
trySplitWords 40
val 41
val1 41
validateObject 42
warningp 43
zipit 43

Index **45**

---

add *Add Two Objects*

Description

This function adds two objects. If both objects are numeric vectors, it performs element-wise addition. If one or both objects are strings, it concatenates them. For other objects, it attempts to use the + method defined for the class of the objects.

Usage

```
add(x, y)
```

Arguments

x	An object.
y	An object.

Value

The result of adding the two objects.

Examples

```
# Add two numeric vectors
add(c(1, 2, 3), c(4, 5, 6))
# Concatenate two strings
add("hello", "world")
# Add a number and a string (concatenation)
add(1, " world")
```

bound *Bound a Number within a Range*

Description

This function bounds a number within a specified range.

Usage

```
bound(num, lowerbound, upperbound)
```

Arguments

num	A numeric vector to be bounded.
lowerbound	The lower bound of the range.
upperbound	The upper bound of the range.

Value

A numeric vector with elements bounded within the specified range.

Examples

```
bound(1, 0, 2)
bound(1:10, -1, 5)
```

closestWord *Find the Closest Word in a Set to a Given Word*

Description

This function finds the closest word in a set of words to a given word based on a specified distance function.

Usage

```
closestWord(s, strset, distFunc = utils::adist)
```

Arguments

s	A character string.
strset	A set of character strings.
distFunc	A function to compute distance between strings. Default is utils::adist.

Value

The closest word in the set to the given word.

Examples

```
# Find the closest word to "hello" in the set c("hallo", "hullo", "hey")
closestWord("hello", c("hallo", "hullo", "hey"))
```

`colToRownames` *Convert a Column to Row Names*

Description

This function converts a specified column of a data frame to row names, ensuring uniqueness if necessary.

Usage

```
colToRownames(df, col, .remove = TRUE, .uniqueSep = ".")
```

Arguments

`df`	A data frame.
`col`	A character string specifying the name of the column to convert to row names.
`.remove`	A logical indicating whether to remove the selected column after converting to row names. Default is TRUE.
`.uniqueSep`	A character string to separate duplicate row names when ensuring uniqueness with `make.unique()`. Default is ".".

Value

A data frame with the specified column as row names. If `.remove` is TRUE, the original column is removed.

See Also

`mutateToRownames()`

Examples

```
# Convert the 'ID' column to row names
df <- data.frame(ID = c("A", "B", "C"), Value = c(10, 20, 30))
colToRownames(df, "ID")
# Convert the 'ID' column to row names and keep the column
df <- data.frame(ID = c("A", "B", "C"), Value = c(10, 20, 30))
colToRownames(df, "ID", .remove = FALSE)
```

createHash *Create a Hash Table*

Description

This function creates a hash table from a set of keys and optional initial value. Note that it is simply a convenience wrapper for the hash package.

Usage

```
createHash(keys, init_vals = NULL)
```

Arguments

keys	A vector of keys for the hash table.
init_vals	Optional initial value for the hash table.

Value

A hash table with the specified keys and initial values.

Examples

```
# Create a hash table with keys and no initial values
createHash(c("a", "b", "c"))
# Create a hash table with keys and initial value of 0
createHash(c("a", "b", "c"), 0)
```

createMutator *Create a Mutator Function*

Description

This function creates a mutator function based on a specified binary operator. The output mutator function updates a variable in the parent frame by applying the binary operator with a given value. It's recommended to use this function to easily construct special functions in the form of %f% where f can be any symbol of choice. See examples.

Usage

```
createMutator(binaryOperator)
```

Arguments

binaryOperator	A binary operator function to apply for the mutation.

Value

A function that takes a variable and a value, applying the binary operator to update the variable in the parent frame.

Examples

```
"%+=%" <- createMutator(add)
x <- 1
x %+=% 1
x # becomes 2
```

`createPkgLoader` *Create Package Loader Function*

Description

[Experimental]

This function creates a package loader function that can install and load packages from CRAN, Bioconductor, or GitHub, optionally displaying verbose output. This function can be useful in new R instances with little dependencies available.

The function takes the following arguments:

- `cran`
- `bioc`
- `gh`
- `verbose`

where cranandbioctake `character vectors of package names on CRAN and Bioconductor, whileghtakes character` verbose‘ takes in a logical for whether to display additional informative messages in the REPL.

The function will not install packages that can already be loaded by default.

Usage

```
createPkgLoader(lib = .libPaths()[1])
```

Arguments

`lib`	A character vector specifying the library directory for package installation of the output function. Defaults to the current default package installation directory in `.libPaths()[1]`

Value

A function that installs and loads packages.

Examples

```
# Create the package loader function
loader <- createPkgLoader()

# # commented usage example
# loader(
#   cran = c("dplyr", "ggplot2"),
#   bioc = c("GenomicRanges", "Biobase"),
#   gh = c("tidyverse/dplyr"),
#   verbose = FALSE
# )
```

`divide` *Divide Two Numbers*

Description

This function divides one number by another.

Usage

```
divide(x, y)
```

Arguments

`x`	A numeric vector.
`y`	A numeric vector.

Value

A numeric vector representing the quotient of the input vectors.

Examples

```
# Divide two numeric vectors
divide(c(10, 20, 30), c(2, 4, 5))
```

enclose *Enclose String with Specified Characters*

Description

This function encloses a string with specified characters on the left and the right.

Usage

```
enclose(x, left, right)
```

Arguments

x	A character string to enclose.
left	A character string to prepend.
right	A character string to append.

Value

A new character string with x enclosed by left and right.

Examples

```
enclose("text", "[", "]") # returns "[text]"
```

encloseBr *Enclose String with Brackets*

Description

This function encloses a string with parentheses.

Usage

```
encloseBr(x)
```

Arguments

x	A character string to enclose.

Value

A new character string with x enclosed by parentheses.

Examples

```
encloseBr("text") # returns "(text)"
```

enumerateit *Enumerate Elements with Indices*

Description

This function pairs elements of vectors or lists with their indices. The output is meant to be used in a for loop, and each element extracted with the `ind()`, `val()`, or `val1()` functions. A slightly lighter weight alternative to `itertools::enumerate()`

Usage

```
enumerateit(..., zero_indexed = FALSE)
```

Arguments

`...`	Vectors or lists to be enumerated.
`zero_indexed`	A logical indicating whether indexing should start from zero. Default is FALSE.

Value

A list of lists, where each inner list contains an index and the corresponding elements from the input vectors or lists.

See Also

`ind()`, `val()`, `val1()`

Examples

```
# Enumerate a vector
enumerateit(c("a", "b", "c"))
# Enumerate a vector starting from zero
enumerateit(c("a", "b", "c"), zero_indexed = TRUE)
# Enumerate two vectors
enumerateit(c(1, 2), c("x", "y"))
```

findMissingRdSections *Find Missing Sections in Rd Files*

Description

[Experimental]

This function scans Rd documentation files in the specified package's `\man` directory to identify which functions lack certain documentation sections like `\examples`. If there are no missing sections in all the Rd files, then the output is a `character(0)`

Usage

```
findMissingRdSections(
  sectionName,
  pkg = ".",
  ignore = NULL,
  .ignore = "-package$"
)

fmrs(sectionName, pkg = ".", ignore = NULL, .ignore = "-package$")
```

Arguments

sectionName	A character vector of the Rd sections to look for.
pkg	The path to the package directory, defaulting to the current directory ".".
ignore	Additional Regexes of *function names* to be ignored in the output.
.ignore	More regexes of functions to ignore set by default. Will be appended with the ignore regexes and unioned with joinRegex().

Value

Character vector of function names that are missing any of the specified sections in their Rd files. May be length 0 if all fulfill criteria.

Examples

```
try(
  findMissingRdSections(c("examples", "example"), pkg = "."),
  silent = TRUE
)
```

---

fixColnames *Fix Column Names*

Description

[Experimental]

This function fixes the column names of a given object so that all words are spaced by a specified delimiter, and any special characters are replaced according to a substitution map.

Usage

```
fixColnames(
  object,
  invalidRegex = "( )|(\\()|(\\))|(\\.)|(/)",
  spacing = "_",
```

```
  subMap = NULL,
 .subMap = list(`%+` = "pct", `\\$+` = "dollars", `\\++` = "plus", `-+` = "minus",
    `\\*+` = "star", `#+` = "cnt", `&+` = "and", `@+` = "at"),
  unique = FALSE
)
```

Arguments

object	A data frame or matrix.
invalidRegex	A character string containing a regular expression pattern for invalid characters to replace. Default is "( )|(\()|(\))|(\.)|(/)".
spacing	A character string to replace invalid characters with. Default is "_".
subMap	A named list where the names are regular expressions and the values are the replacement strings. These substitutions are applied before .subMap.
.subMap	A named list where the names are regular expressions and the values are the replacement strings. These substitutions are applied after subMap. Default is list("\+" = "plus").
unique	A logical indicating whether to ensure unique column names by appending a suffix if necessary. Default is FALSE.

Value

The data frame or matrix with fixed column names.

Examples

```
# Fix column names of a data frame
df <- data.frame(`A (1)` = c(1, 2, 3), `B/C` = c(4, 5, 6), `D+E` = c(7, 8, 9))
fixColnames(df)
```

---

getAvgHex *Compute the Average of Hex Colors*

Description

This function computes the average color of the provided hex color values.

Usage

```
getAvgHex(...)
```

Arguments

...	Hex color values as character strings. Could also be any number of character vectors (including lists) which will all be coerced into a single characters, assuming they are valid hex codes.

Value

A single hex color character representing the average of the input colors.

Source

https://stackoverflow.com/questions/649454

Examples

```
getAvgHex("#000000", "#FF00FF")
getAvgHex(c("#008040", "#000000", "#FF00FF"))

# very nonstandard but possible way to input hexes. Essentially,
# any combination of vectors will work.
getAvgHex(list("#008040", "#000000"), "#FF00FF", c("#FF00FF"))
```

---

getChar *Get a Character at a Specific Index*

Description

This function retrieves a character at a specific index from a string.

Usage

```
getChar(x, index)
```

Arguments

x	A character string.
index	The index of the character to retrieve.

Value

The character at the specified index.

Examples

```
# Get the character at index 2
getChar("hello", 2)
```

getfirst *Get the First Elements of a Vector or List*

Description

This function retrieves the first n elements of a vector or list.

Usage

```
getfirst(x, n = 1)

## Default S3 method:
getfirst(x, n = 1)
```

Arguments

x	A vector, list, or other supported data type.
n	An integer specifying the number of elements to retrieve from the start. Default is 1.

Value

The first n elements of the input.

Examples

```
# Get the first element of a vector
getfirst(c(1, 2, 3, 4, 5))
# Get the first 2 elements of a vector
getfirst(c(1, 2, 3, 4, 5), 2)
# Get the first element of a list
getfirst(list("a", "b", "c"))
# Get the first 2 elements of a list
getfirst(list("a", "b", "c"), 2)
```

getlast *Get the Last Elements of a Vector or List*

Description

This function retrieves the last n elements of a vector or list.

Usage

```
getlast(x, n = 1)

## Default S3 method:
getlast(x, n = 1)
```

Arguments

x	A vector, list, or other supported data type.
n	An integer specifying the number of elements to retrieve from the end. Default is 1.

Value

The last n elements of the input.

Examples

```
# Get the last element of a vector
getlast(c(1, 2, 3, 4, 5))
# Get the last 2 elements of a vector
getlast(c(1, 2, 3, 4, 5), 2)
# Get the last element of a list
getlast(list("a", "b", "c"))
# Get the last 2 elements of a list
getlast(list("a", "b", "c"), 2)
```

---

getPkgKeywords *Get Keywords from R Package Documentation*

Description

[Experimental]

This function retrieves keywords from all package documentation files located in the /man directory of the specified R package. It can return a unique list of keywords or a frequency distribution of these keywords as a table object, sorted by the keys.

Note that the "internal" keyword is ignored.

Usage

```
getPkgKeywords(pkg = ".", asDistribution = FALSE)
```

Arguments

pkg	The path to the R package directory.
asDistribution	Logical; if FALSE, returns a character vector of unique keywords. If TRUE, returns a table with the frequency of each keyword.

Value

If asDistribution is FALSE, a sorted character vector of unique keywords is returned. If asDistribution is TRUE, a table of keywords and their frequencies is returned. If no keywords were detected, returns a character of length 0.

Examples

```
getPkgKeywords()
getPkgKeywords(asDistribution = TRUE)
```

`getPlotDims` *Get the xmin, xmax, ymin, ymax of a ggplot Object*

Description

This function retrieves the minimum and maximum x and y dimensions of a ggplot object. Note that it is the dimension of the plot within the x and y axis and not the dimensions of the actual output image itself. This may be useful for numerical computations when modifying plots, but can be slow since it builds the actual plot first.

Usage

```
getPlotDims(plt)
```

Arguments

`plt`	A ggplot object.

Value

A list with elements `xr` (a vector of xmin and xmax) and `yr` (a vector of ymin and ymax).

Examples

```
library(ggplot2)
getPlotDims(ggplot(mtcars) + geom_point(aes(mpg, cyl)))
```

`getUniquePairsUpTo` *Generate Unique Pairs Up To a Number*

Description

This function generates all unique pairs of integers up to a given number.

Usage

```
getUniquePairsUpTo(x, oneIndexed = TRUE)
```

Arguments

x	An integer specifying the upper limit for pairs.
oneIndexed	A logical indicating whether the pairs should be one-indexed. Default is TRUE.

Value

A list of unique pairs of integers up to the specified number.

Examples

```
# Generate unique pairs up to 3 (one-indexed)
getUniquePairsUpTo(3)
# Generate unique pairs up to 3 (zero-indexed)
getUniquePairsUpTo(3, oneIndexed = FALSE)
```

---

greplDir *Search for a Pattern in Files within a Directory*

Description

[Experimental]

The `greplDir` function searches for a specified pattern in all files within a given directory. It allows for optional exclusion of files matching a specified regular expression. Note that all files are assumed to be a single string, with each line joined by the newline character `"\n"`.

Usage

```
greplDir(fpattern, dirPath = getwd(), fIgnoreRegex = NULL, ...)
```

Arguments

fpattern	Character. The pattern to search for within the files.
dirPath	Character. The path to the directory containing files to be searched.
fIgnoreRegex	Character. A regular expression to match file names that should be ignored (default is NULL).
...	Additional arguments passed to `listFiles()`, which are passed to `list.files()`

Value

A named logical vector indicating which files contain the pattern. The names attribute contains the file names.

Examples

```
result <- tryCatch(
  greplDir("error", fIgnoreRegex = "\\.log$"),
  warning = function(w) c(exFname = TRUE),
  error = function(e) c(exFname = TRUE)
)
# its even more useful to use `base::which` on the result to
# get matches and mismatches - this returns it with names
# by default
which(result)
which(!result)
```

---

ind *Get Index from Enumerated Element*

Description

This function extracts the index from an enumerated element.

Usage

```
ind(elem)
```

Arguments

elem	An enumerated element.

Value

The index of the enumerated element.

See Also

enumerateit()

Examples

```
# Extract index from an enumerated element
elem <- list(1, "a")
ind(elem)
```

initDataFrameWithColnames *Initialize a DataFrame with Column Names*

Description

This function creates an empty data frame and assigns the specified column names with zero rows.

Usage

```
initDataFrameWithColnames(colnames = NULL)
```

Arguments

colnames	A character vector specifying the names of the columns for the data frame. This vector will be attempted to be coerced to a character.

Value

A data frame with the specified column names. Non unique names will be handled by the conventions of `data.frame()`. prefixes.

Examples

```
# Create a data frame with specified column names initialized with NA
initDataFrameWithColnames(c("Name", "Age", "Gender"))
```

initEmptyTable *Initialize an Empty Table*

Description

This function initializes an empty table.

Usage

```
initEmptyTable()
```

Value

An empty table structure.

Examples

```
# Create an empty table
initEmptyTable()
```

initList *Initialize a List*

Description

This function initializes a list based on size or names, with an optional initial value.

Usage

```
initList(x = NULL, initVal = NULL)
```

Arguments

x	A character vector as names, or an numeric indicating the size of the list.
initVal	an optional initial value for all elements of the list.

Value

A list of the specified size and names, optionally initialized with a value.

Examples

```
# Create a list with 3 elements
initList(3)
# Create a named list initialized with NULL
initList(c("a", "b", "c"))
# Create a list with 2 elements initialized with 0
initList(2, 0)
```

initTestthat *Initialize Testthat Files*

Description

This function scans all files in the specified R directory based on its name, excluding some based on the patterns provided in the `ignore` argument, and creates `testthat` files if they are missing. Useful for when many source code files were created from rapid development and unit testing has yet to be setup.

Usage

```
initTestthat(
  rDir = "R",
  testDir = "tests/testthat",
  .ignore = c("-package.R$", "-class.R$", "^data.R$", "^zzz.R$", "^RcppExports.R$"),
  ignore = NULL
)
```

Arguments

rDir	The directory containing R source files. Default is "R".
testDir	The directory where testthat files should be created. Default is "tests/testthat".
.ignore	A character vector specifying regex patterns of files to ignore. Defaults to common patterns c("-package.R$", "-class.R$", "^data.R$", "^zzz.R$", "^RcppExports.R$")
ignore	A character vector of extra regex patterns of R files to ignore

Value

No return value, called for side effects.

Examples

```
try({

initTestthat()
initTestthat(rDir = "src", testDir = "tests")
initTestthat(ignore = c("^foo", "-bar.R$"))

}, silent = TRUE)
```

---

initV *Initialize a Vector*

Description

This function initializes a vector based on a specified type and size, with an optional initial value.

Usage

```
initV(typeFunc, x, initVal = NULL)
```

Arguments

typeFunc	A character string indicating the type of the vector or a function to create the vector.
x	The length of the vector.
initVal	An optional initial value to fill the vector.

Value

A vector of the specified type and size, optionally initialized with a value.

Examples

```
# Create a numeric vector of length 5
initV("numeric", 5)
# Create a logical vector of length 3 initialized with TRUE
initV("logical", 3, TRUE)
```

`isBound` *Check if a Number is within a Range*

Description

This function checks if a number is within a specified range.

Usage

```
isBound(num, lowerbound, upperbound)
```

Arguments

`num`	A numeric vector to be checked.
`lowerbound`	The lower bound of the range.
`upperbound`	The upper bound of the range.

Value

A logical vector indicating whether each element is within the specified range.

Examples

```
isBound(1, 0, 2)
isBound(1:10, -1, 5)
```

`isCamelCase` *Check if String is camelCase*

Description

This function checks if a given string adheres to camelCase naming conventions, starting with a lowercase letter followed by any combination of upper and lower case letters.

Usage

```
isCamelCase(x)
```

Arguments

x	A character string to check.

Value

TRUE if the string is camelCase, FALSE otherwise.

Examples

```
isCamelCase("camelCase")   # returns TRUE
isCamelCase("CamelCase")   # returns FALSE
isCamelCase("camelcase")   # returns TRUE
```

---

isEven *Check if a Number is Even*

Description

This function checks if a number is even.

Usage

```
isEven(x)
```

Arguments

x	A numeric vector.

Value

A logical vector indicating whether each element is even.

Examples

```
# Check if numbers are even
isEven(c(1, 2, 3, 4))
```

isOdd *Check if a Number is Odd*

Description

This function checks if a number is odd.

Usage

```
isOdd(x)
```

Arguments

x	A numeric vector.

Value

A logical vector indicating whether each element is odd.

Examples

```
# Check if numbers are odd
isOdd(c(1, 2, 3, 4))
```

isPascalCase *Check if String is PascalCase*

Description

This function checks if a given string adheres to PascalCase naming conventions, starting with an uppercase letter followed by any combination of upper and lower case letters.

Usage

```
isPascalCase(x)
```

Arguments

x	A character string to check.

Value

TRUE if the string is PascalCase, FALSE otherwise.

Examples

```
isPascalCase("PascalCase") # returns TRUE
isPascalCase("pascalCase") # returns FALSE
isPascalCase("Pascalcase") # returns FALSE
```

isSnakeCase *Check if String is snake_case*

Description

This function checks if a given string adheres to snake_case naming conventions. By default (strict = TRUE), it only allows lowercase letters separated by underscores. If strict is FALSE, uppercase letters are also permitted.

Usage

```
isSnakeCase(x, strict = TRUE)
```

Arguments

x	A character string to check.
strict	Logical indicating whether the string should strictly contain only lowercase letters (TRUE) or can include uppercase letters (FALSE). Default is TRUE.

Value

TRUE if the string is snake_case according to the specified strictness, FALSE otherwise.

Examples

```
isSnakeCase("snake_case")        # returns TRUE
isSnakeCase("Snake_Case")        # returns FALSE
isSnakeCase("snake_case", FALSE) # returns TRUE
isSnakeCase("Snake_Case", FALSE) # returns TRUE
```

isVowel *Check if a Character is a Vowel*

Description

This function checks if a character is a vowel.

Usage

```
isVowel(x)
```

Arguments

x	A character.

Value

TRUE if the character is a vowel, FALSE otherwise.

Examples

```
# Check if 'a' is a vowel
isVowel("a")
# Check if 'b' is a vowel
isVowel("b")
```

`joinRegex` *Join regex expressions by union*

Description

This function simply joins a vector of regex characters by union, and produces a single character regex in the form of `(foo)|(bar)`.

Usage

```
joinRegex(...)
```

Arguments

`...`	character vectors of the regex expressions to join. Both vectors and individual characters of any length will work

Value

a character of the unioned regex

Examples

```
joinRegex(c("^foo", "bar$"))
joinRegex("^foo", "bar$", "[bB]az")
```

listFiles *List Only Files in a Directory*

Description

This function lists only the files in a specified directory, excluding directories. It is useful when you need to process or analyze only the files within a directory without including subdirectories. The `base::list.files()` function lists both files and directories, so this function provides a more convenient way to obtain just the files.

Usage

```
listFiles(dirPath, ...)
```

Arguments

`dirPath`	Character. The path to the directory from which to list files.
`...`	Additional arguments passed to `base::list.files()` (e.g., `pattern`, `recursive`). Note that `full.names` will be ignored.

Value

A character vector of file paths.

Examples

```
listFiles(getwd())
listFiles(getwd(), pattern = "\\.R$", recursive = TRUE)
```

multiply *Multiply Two Numbers*

Description

This function multiplies two numbers.

Usage

```
multiply(x, y)
```

Arguments

`x`	A numeric vector.
`y`	A numeric vector.

Value

A numeric vector representing the product of the input vectors.

Examples

```
# Multiply two numeric vectors
multiply(c(2, 3, 4), c(5, 6, 7))
```

`mutateToRownames` *Mutate columns to Row Names*

Description

[Experimental]

This function sets new row names for a data frame based on a tidy evaluation expression.

Usage

```
mutateToRownames(.data, expr, .remove = FALSE, .uniqueSep = ".")
```

Arguments

`.data`	A data frame.
`expr`	A tidy evaluation expression specifying the columns to use for the new row names.
`.remove`	A logical indicating whether to remove the selected columns. Default is FALSE.
`.uniqueSep`	A character string to separate duplicate row names when ensuring uniqueness with `make.unique()`. Default is ".".

Value

A data frame with updated row names.

Examples

```
library(dplyr)

mtcars %>%
    head() %>%
    mutateToRownames(wt + 3*vs)
```

namedNumericToTable *Convert Named Numeric Vector to Table*

Description

This function converts a named numeric vector to a table.

Usage

```
namedNumericToTable(x)
```

Arguments

x	A named numeric vector.

Value

A table with the same names and values as the input vector.

Examples

```
# Convert a named numeric vector to a table
vec <- c(a = 1, b = 2, c = 3)
namedNumericToTable(vec)
```

prependIndefArticle *Prepend an Indefinite Article to a String*

Description

This function prepends an indefinite article ("a" or "an") to a string based on whether it starts with a vowel or not.

Usage

```
prependIndefArticle(x)
```

Arguments

x	A character string.

Value

The string with an indefinite article prepended.

Examples

```
# Prepend an indefinite article to "apple"
prependIndefArticle("apple")
# Prepend an indefinite article to "banana"
prependIndefArticle("banana")
```

---

quietly *Suppress Messages and Output*

Description

This function suppresses messages and captures output from an expression.

Usage

```
quietly(e)
```

Arguments

e	An expression to evaluate.

Value

The result of the expression with messages suppressed and output captured.

Examples

```
quietly(print(1))

quietly({
    print(1)
    print(2)
    print(3)
})

a <- 1
quietly({
    a <- a + 1
    print(a)
})
```

rmByName *Remove Elements with Specified Name Regex*

Description

This function removes elements from an indexable object (e.g., a named vector or list) where the names match a specified regular expression.

Usage

```
rmByName(x, pattern, silent = FALSE)
```

Arguments

x	An indexable object (e.g., a named vector, list, or data frame).
pattern	A character containing a regular expression(s) to match the names of elements to be removed.
silent	A logical indicating whether to silence a warning if no names are detected.

Value

The input object with elements removed based on the name regex.

Examples

```
myList <- list(a = 1, b_test = 2, c = 3, d_test = 4)
rmByName(myList, "_test")
```

rownamesToCol *Convert Row Names to a Column*

Description

This function converts the row names of a data frame to a specified column. Note that if the specified column already exists, it is overwritten.

Usage

```
rownamesToCol(df, colname = "rownames")
```

Arguments

df	A data frame.
colname	A character string specifying the name of the new column to contain the row names. Defaults to "rownames".

Value

A data frame with the row names converted to a column.

Examples

```
# Convert row names to a column named 'ID'
df <- data.frame(Value = c(10, 20, 30))
rownames(df) <- c("A", "B", "C")
rownamesToCol(df, "ID")
```

---

scaleHex *Scale the Brightness of a Hex Color*

Description

This function scales the brightness of hex colors by a given factor.

Usage

```
scaleHex(hex, scaleFactor)
```

Arguments

hex	Hex color values as characters.
scaleFactor	A numeric value to scale the brightness. A value of 1 returns the original color.

Value

A hex color value with adjusted brightness.

Examples

```
scaleHex("#404040", 2)
```

setColnames *Set Column Names*

Description

This function sets new column names for a given data frame or matrix.

Usage

```
setColnames(object, newColnames)
```

Arguments

object	A data frame or matrix.
newColnames	A character vector specifying the new column names.

Value

The data frame or matrix with updated column names.

Examples

```
# Set new column names for a data frame
df <- data.frame(A = c(1, 2, 3), B = c(4, 5, 6))
setColnames(df, c("X", "Y"))
```

setRownames *Set Row Names*

Description

This function sets new row names for a given data frame or matrix.

Usage

```
setRownames(object, newRownames)
```

Arguments

object	A data frame or matrix.
newRownames	A character vector specifying the new row names.

Value

The data frame or matrix with updated row names.

Examples

```
# Set new row names for a data frame
df <- data.frame(A = c(1, 2, 3), B = c(4, 5, 6))
setRownames(df, c("row1", "row2", "row3"))
```

`splitCamel` *Split CamelCase or PascalCase Strings*

Description

This function splits strings formatted in camelCase or PascalCase into their component words. It can handle words where uppercase letters transition to lowercase letters, and it is capable of handling strings with sequences of uppercase letters followed by lowercase letters, effectively separating acronyms from camelCase beginnings.

Usage

```
splitCamel(x, conseq = TRUE)

splitPascal(x, conseq = TRUE)
```

Arguments

`x`	A character vector containing CamelCase or PascalCase strings to be split.
`conseq`	Logical indicating whether consecutive uppercase letters should be treated as part of the previous word (TRUE) or as separate words (FALSE). Default is TRUE.

Value

A list of character vectors, each containing the parts of the corresponding CamelCase or PascalCase string split at the appropriate transitions. If `conseq` is FALSE, acronyms followed by words are separated.

Source

https://stackoverflow.com/questions/8406974/splitting-camelcase-in-r

Examples

```
splitCamel("splitCamelCaseIntoWords")
splitCamel(c("fooBar", "FOOBar", "anotherFOOBarTest"), conseq = FALSE)
```

splitSnake *Split Snake Case String*

Description

This function splits a string formatted in snake_case into its component words, using underscores as delimiters. It is useful for parsing identifiers or variable names that follow snake_case naming conventions.

Usage

```
splitSnake(x)
```

Arguments

x	A character string in snake_case to be split.

Value

A list of character vectors, each containing the parts of the string split at underscores.

Examples

```
splitSnake("this_is_snake_case")
splitSnake("another_example_here")
```

startsWithVowel *Check if a String Starts with a Vowel*

Description

This function checks if a string starts with a vowel.

Usage

```
startsWithVowel(x)
```

Arguments

x	A character string.

Value

TRUE if the string starts with a vowel, FALSE otherwise.

Examples

```
# Check if "apple" starts with a vowel
startsWithVowel("apple")
# Check if "banana" starts with a vowel
startsWithVowel("banana")
```

`stopp` *Custom Stop Function Without Call*

Description

This function provides a wrapper around the base `stop` function, but it automatically sets `call.` to FALSE, which means the function call itself is not included in the resulting error message. This makes error messages cleaner. The `domain` argument can be used to specify a translation domain.

Usage

```
stopp(..., domain = NULL)
```

Arguments

`...`	Arguments passed on to `stop`.
`domain`	The translation domain, NULL by default.

Value

No return value, this function stops execution of the program.

See Also

`stop()`

Examples

```
try(stopp("This is a custom stop message without the call."), silent = TRUE)
```

stripSpaces *Remove Spaces from a String*

Description

This function removes spaces from a character string.

Usage

```
stripSpaces(x)
```

Arguments

x	A character string.

Value

The string with spaces removed.

Examples

```
# Remove spaces from "hello world"
stripSpaces("hello world")
```

substrEnd *Extract Substring from Start to End Difference*

Description

This function extracts a substring from a given start position to the position determined by subtracting `endDiff` from the string length.

Usage

```
substrEnd(x, start, endDiff)
```

Arguments

x	A character string from which the substring is extracted.
start	The starting position for the substring extraction.
endDiff	The difference to subtract from the string length to determine the end position.

Value

A substring of the input character string.

See Also

substr()

Examples

```
substrEnd("12345", 1, 1)
substrEnd("12345", 1, 2)
substrEnd("12345", 2, 3)
```

---

subtract *Subtract Two Numbers*

Description

This function subtracts one number from another.

Usage

```
subtract(x, y)
```

Arguments

x	A numeric vector.
y	A numeric vector.

Value

A numeric vector representing the difference between the input vectors.

Examples

```
# Subtract two numeric vectors
subtract(c(10, 20, 30), c(1, 2, 3))
```

tableToNumeric *Convert a Table to Numeric*

Description

This function converts a table to a numeric vector.

Usage

```
tableToNumeric(x)
```

Arguments

x	A table to be converted.

Value

A numeric vector with names preserved from the table.

Examples

```
# Convert a table to numeric
tbl <- table(c("a", "b", "a"))
tableToNumeric(tbl)
```

test_quietly_that *Run a Testthat test Quietly*

Description

This function runs a `test_that` block quietly, suppressing messages and output from any verbose functions.

Usage

```
test_quietly_that(desc, code)
```

Arguments

desc	A description of the test.
code	The code to be tested.

Value

No return value, called for side effects.

Examples

```
# Run a test quietly
test_quietly_that("quiet test example", {
    testthat::expect_equal(1 + 1, 2)
})
```

`trySplitWords` *Try to Split Words Based on Naming Convention*

Description

This function attempts to split characters into its component words (and by default, all in lowercase) based on camelCase, PascalCase, or snake_case conventions. If the string does not match any of these conventions, it returns all groups of letters.

Usage

```
trySplitWords(..., conseq = TRUE, strictSnake = FALSE, uncase = TRUE)
```

Arguments

`...`	character(s) to be split, treated as a single vector after unlisting
`conseq`	A logical indicating whether the `conseq` argument in `splitCamel()`/`splitPascal()` should be `TRUE` or `FALSE`.
`strictSnake`	A logical indicating the `strict` argument in `isSnakeCase()`.
`uncase`	A logical indicating whether to remove all casing in the output to lowercase.

Value

A list of character vectors, each containing the parts of the string split into individual words.

See Also

`splitCamel`, `splitPascal`, `splitSnake`, `isCamelCase`, `isPascalCase`, `isSnakeCase`

Examples

```
trySplitWords("camelCaseExample")
trySplitWords("PascalCaseExample")
trySplitWords("snake_case_example", c("more_snake_cases"), "third_snake_case")
trySplitWords("some|random|case")
trySplitWords("Space Words", "UPPER_CASE", uncase = TRUE)
```

val *Get Value from Enumerated Element by Index*

Description

This function extracts the value from an enumerated element by the given index.

Usage

```
val(elem, index)
```

Arguments

elem	An enumerated element.
index	The index of the value to extract.

Value

The value at the specified index in the enumerated element.

See Also

enumerateit()

Examples

```
# Extract value from an enumerated element by index
elem <- list(1, "a", "b")
val(elem, 2)
```

val1 *Get First Value from Enumerated Element*

Description

This function extracts the first value from an enumerated element.

Usage

```
val1(elem)
```

Arguments

elem	An enumerated element.

Value

The first value in the enumerated element.

See Also

enumerateit()

Examples

```
# Extract the first value from an enumerated element
elem <- list(1, "a", "b")
val1(elem)
```

---

validateObject *Validate Object*

Description

This function validates an object using a list of checks. If any check fails, an error handler is called and a default value is returned.

Usage

```
validateObject(obj, checks, errorHandler = warningp, defaultReturn = NULL)
```

Arguments

obj	The object to validate.
checks	A list of functions, each taking the object as an argument and returning NULL if the check passes or an error message if the check fails.
errorHandler	A function to handle errors, taking the error message as an argument. Default is warning.
defaultReturn	The value to return if any check fails. Default is NULL.

Value

The original object if all checks pass, or defaultReturn if any check fails.

Examples

```
# Define some checks
checkNotNull <- function(x) if (is.null(x)) "Object is NULL" else NULL
checkIsNumeric <- function(x) if (!is.numeric(x)) "Object is not numeric" else NULL

# Validate an object
obj <- 42
validateObject(obj, list(checkNotNull, checkIsNumeric))
```

```
# Validate an object that fails a check
obj <- NULL
try(validateObject(obj, list(checkNotNull, checkIsNumeric), errorHandler = stop), silent = TRUE)
```

`warningp` *Custom Warning Function Without Call*

Description

This function provides a wrapper around the base `warning` function, adding flexibility to warnings by setting `call.` to FALSE automatically. This modification means that the function call is not included in the warning message, streamlining the output for users.

Usage

```
warningp(...)
```

Arguments

`...`	Arguments passed on to `warning`.

Value

No return value, this function issues a warning.

See Also

`warning`

Examples

```
try(warningp("This is a custom warning message without the call."), silent = TRUE)
```

`zipit` *Zip Multiple Vectors or Lists*

Description

This function combines multiple vectors or lists element-wise into a list of lists. A slightly lighter weight alternative to `itertools::izip()`

Usage

```
zipit(...)
```

Arguments

...	Vectors or lists to be combined.

Value

A list of lists, where each inner list contains the elements from the corresponding positions in the input vectors or lists.

See Also

enumerateit()

Examples

```
# Zip two vectors
zipit(c(1, 2, 3), c("a", "b", "c"))
# Zip three vectors
zipit(c(1, 2), c("x", "y"), c(TRUE, FALSE))
```

Index

∗ **character**
enclose, 9
encloseBr, 9
getChar, 13
substrEnd, 37
∗ **color**
getAvgHex, 12
scaleHex, 32
∗ **dataInitialization**
createHash, 6
initDataFrameWithColnames, 19
initEmptyTable, 19
initList, 20
initV, 21
namedNumericToTable, 29
tableToNumeric, 39
∗ **fileSystem**
listFiles, 27
∗ **ggplot**
getPlotDims, 16
∗ **higherOrderFunctions**
createMutator, 6
∗ **indexing**
getfirst, 14
getlast, 14
∗ **interactivity**
quietly, 30
∗ **iteration**
enumerateit, 10
getUniquePairsUpTo, 16
ind, 18
val, 41
val1, 41
zipit, 43
∗ **math**
add, 3
bound, 4
divide, 8
isBound, 22
isEven, 23
isOdd, 24
multiply, 27
subtract, 38
∗ **packageDevelopment**
findMissingRdSections, 10
getPkgKeywords, 15
∗ **packageLoading**
createPkgLoader, 7
∗ **regex**
greplDir, 17
joinRegex, 26
rmByName, 31
∗ **spelling**
closestWord, 4
isCamelCase, 22
isPascalCase, 24
isSnakeCase, 25
isVowel, 25
prependIndefArticle, 29
splitCamel, 34
splitSnake, 35
startsWithVowel, 35
stripSpaces, 37
trySplitWords, 40
∗ **testing**
initTestthat, 20
test_quietly_that, 39
∗ **validation**
stopp, 36
validateObject, 42
warningp, 43
∗ **wrangling**
colToRownames, 5
fixColnames, 11
mutateToRownames, 28
rownamesToCol, 31
setColnames, 33
setRownames, 33

add, 3

base::list.files(), *27*
bound, 4

closestWord, 4
colToRownames, 5
createHash, 6
createMutator, 6
createPkgLoader, 7

data.frame(), *19*
divide, 8

enclose, 9
encloseBr, 9
enumerateit, 10
enumerateit(), *18*, *41*, *42*, *44*

findMissingRdSections, 10
fixColnames, 11
fmrs *(*findMissingRdSections*)*, 10

getAvgHex, 12
getChar, 13
getfirst, 14
getlast, 14
getPkgKeywords, 15
getPlotDims, 16
getUniquePairsUpTo, 16
greplDir, 17

ind, 18
ind(), *10*
initDataFrameWithColnames, 19
initEmptyTable, 19
initList, 20
initTestthat, 20
initV, 21
isBound, 22
isCamelCase, 22, *40*
isEven, 23
isOdd, 24
isPascalCase, 24, *40*
isSnakeCase, 25, *40*
isSnakeCase(), *40*
isVowel, 25

joinRegex, 26
joinRegex(), *11*

list.files(), *17*
listFiles, 27
listFiles(), *17*

make.unique(), *5*, *28*
multiply, 27
mutateToRownames, 28
mutateToRownames(), *5*

namedNumericToTable, 29

prependIndefArticle, 29

quietly, 30

rmByName, 31
rownamesToCol, 31

scaleHex, 32
setColnames, 33
setRownames, 33
splitCamel, 34, *40*
splitCamel(), *40*
splitPascal, *40*
splitPascal *(*splitCamel*)*, 34
splitPascal(), *40*
splitSnake, 35, *40*
startsWithVowel, 35
stop, *36*
stop(), *36*
stopp, 36
stripSpaces, 37
substr(), *38*
substrEnd, 37
subtract, 38

tableToNumeric, 39
test_quietly_that, 39
trySplitWords, 40

val, 41
val(), *10*
val1, 41
val1(), *10*
validateObject, 42

warning, *43*
warningp, 43

zipit, 43