Package ‘hgutils’

September 7, 2019

Version 0.2.7
Date 2019-09-07
Title Collection of Utility Functions

Description A handy collection of utility functions designed to aid in package development, plotting and scientific research.
Package development functionalities includes among others tools such as cross-referencing package imports with the description file, analysis of redundant package imports, editing of the description file and the creation of package badges for GitHub.
Some of the other functionalities include automatic package installation and loading, plotting points without overlap, creating nice breaks for plots, overview tables and many more handy utility functions.

Depends R (>= 3.2.0)
Imports crayon, dplyr, grDevices, limSolve, magrittr, stats, stringr, utils
Suggests methods, testthat
License GPL-3

URL https://github.com/hvdboorn/hgutils

BugReports https://github.com/hvdboorn/hgutils/issues

Encoding UTF-8
LazyData true
Language en-GB
RoxygenNote 6.1.1
NeedsCompilation no
Author H.G. van den Boorn [aut, cre]
Maintainer H.G. van den Boorn <hvdboorn@gmail.com>
Repository CRAN
Date/Publication 2019-09-07 13:20:03 UTC
R topics documented:

- .pkg_duplicated .................................................. 2
- .regexl ............................................................... 3
- add_badges ........................................................... 3
- analyze_package_imports ........................................... 4
- as.character.patient_flowchart ................................. 5
- create_table_one ................................................. 5
- create_text_table ................................................ 6
- crossref_description ............................................. 7
- description-functions ............................................ 8
- discretize_numbers .............................................. 9
- format_duration .................................................. 10
- frmt ................................................................. 10
- generic_implementations ......................................... 11
- get_breaks .......................................................... 11
- get_square_grid ................................................... 13
- inclusion_flowchart ............................................. 13
- load_packages .................................................... 14
- load_package_collection ......................................... 16
- print.patient_flowchart ......................................... 16
- print.percentage_table .......................................... 17
- progressbar ....................................................... 17
- redundant_packages ............................................. 19
- rm_empty_rows ................................................... 19
- rm_na ............................................................... 20
- rnd_dbl ............................................................. 21
- separate_values .................................................. 21
- sep_thousands .................................................... 22
- spinner ............................................................ 23
- startup ............................................................ 24
- stfu ................................................................. 24
- update_settings .................................................. 25
- valid_pkgname .................................................... 26
- wrap_text_table .................................................. 27

Index 28

---

**.pkg_duplicated**

*Find duplicated packages names*

**Description**

Find duplicated packages names

**Usage**

```
.pkg_duplicated(pkgs)
```
Arguments

pkgs 
A list of packages names

Value

A named list of duplicated names and number of occurrences

Description

Extracts the matches from stringr::str_match[_all]

Usage

.regexl(result)

Arguments

result 
The results from stringr::str_match[_all]

Value

a list of matches

add_badges 

Add badges to the README file for use on Github

Description

Add badges to the README file for use on Github

Usage

add_badges(github_pkg, states = c("active", "abandoned", "concept", 
"inactive", "moved", "suspended", "unsupported", "wip"),
readme_file = "README.md", show_repo_status = TRUE,
show_cran_version = TRUE, show_package_version = TRUE,
show_min_r = TRUE, show_last_update = TRUE, show_travis = TRUE,
show_code_coverage = TRUE)
analyze_package_imports

**Arguments**

- `github_pkg` (The Github repository)
- `states` (Current software cycle state)
- `readme_file` (The filename of the readme file)
- `show_repo_status` (Whether to show the repository status as a badge)
- `show_cran_version` (Whether to show the CRAN version as a badge)
- `show_package_version` (Whether to show the package version as a badge)
- `show_min_r` (Whether to show the minimal R version as a badge)
- `show_last_update` (Whether to show the last update date as a badge)
- `show_travis` (Whether to show the Travis test results as a badge (see https://travis-ci.org))
- `show_code_coverage` (Whether to show the code coverage as a badge (see https://codecov.io))

**Examples**

```r
## Not run:
add_badges("hvdboorn/hgutils")
## End(Not run)
```

**Description**

Analyzes the package imports via `library()` and `load_packages()` in a list of filenames.

**Usage**

```r
analyze_package_imports(files = list.files(pattern = "\.[Rr]$", recursive = TRUE))
```

**Arguments**

- `files` (A vector of filenames of R source files. Typically this is created by `list.files(folder,pattern="\.[Rr]$",recursive = TRUE)`)

**Value**

a named list of results (invisibly). This list contains all import statements, a list of duplicated imports, a list of redundant imports, all function calls in the files with the corresponding imports and a list of packages with the number of function calls.
Examples

```r
## Not run:
analyze_package_imports(list.files(pattern="\.[rR]$", recursive=TRUE))

## End(Not run)
```

---

**as.character.patient_flowchart**

*Text representation of patient inclusion flowchart*

---

**Description**

Text representation of patient inclusion flowchart

**Usage**

```r
## S3 method for class 'patient_flowchart'
as.character(x, length = 7, ...)
```

**Arguments**

- `x`: object to be coerced or tested.
- `length`: Length of the arrows (to the right)
- `...`: further arguments passed to or from other methods.

---

**create_table_one**

*Table one*

---

**Description**

Table one

**Usage**

```r
create_table_one(df, numbers_as_categories = TRUE, deaths = NULL)
create_contingency_table(df, x, max_size = 8,
                          numbers_as_categories = TRUE, ...)
percentage_table(x, n_digits = 2)
```
Arguments

- **df**: data.frame.
- **numbers_as_categories**: Whether numbers should be categorized.
- **deaths**: The number of deaths in the population.
- **x**: column vector name in df.
- **max_size**: maximum size of unique elements in the numeric variable x before the values are clustered.
- **...**: Arguments passed on to get_breaks
  - **limits**: axis limits. May be either a vector of 2 elements with lower and upper bounds, or a single number (which is the upper bound, the lower bound is then assumed to be 0).
  - **N**: step size. The eventual intervals will be multiples of the divisors of N or multiples of N when multiples_only is TRUE. Defaults to 10.
  - **max_breaks**: maximum amount of breaks, defaults to 10.
  - **int_only**: whether only integer divisors of N may be used as breaks, defaults to TRUE.
  - **multiples_only**: whether only multiples of N can be used as breaks, defaults to FALSE.
  - **include_bounds**: whether the resulting breaks should encompass min and max. Defaults to TRUE.
- **n_digits**: The number of digits to which the percentages are rounded.

Value

A dataframe containing the contingency tables for each of the variables in df.
A matrix with distinct (factor) labels and corresponding counts and percentages.

---

create_text_table

Creates a text table

Description

Creates a text table

Usage

create_text_table(string, table_width = 80, compact = TRUE)

Arguments

- **string**: character vector of strings to reformat.
- **table_width**: table character width.
- **compact**: whether to take only the necessary space (TRUE) or to fill out the table_width (FALSE).
crossref_description

Value
A vector of strings per row, forming together a table.

See Also
get_square_grid.

Examples
cat(create_text_table(LETTERS), sep = "\n")

crossref_description  Set imports for DESCRIPTION file

Description
Update the DESCRIPTION file with all imported packages stated in the source code.

Usage
crossref_description(skip_prompt = FALSE, update = TRUE,
                      use_version_numbers = TRUE, rversion = "DEPENDENCIES_VERSION")

Arguments
skip_prompt          whether to skip the confirmation prompt to change the DESCRIPTION file. Defaults to FALSE.
update               whether the DESCRIPTION file should be updated. Defaults to TRUE.
uuse_version_numbers whether package version numbers should be included in the DESCRIPTION file. Defaults to TRUE.
rversion             version of R to be used in the DESCRIPTION file. Can be DEPENDENCIES_VERSION for the latest version in the package dependencies, LATEST_VERSION for the current R version or any valid version number.

Value
Invisibly returns a list with the current R version, the R version obtained from dependencies and packages names (including version numbers).

See Also
numeric_version
Other developer functions: generic_implementations, load_packages, update_settings, valid_pkgname

Examples
## Not run: crossref_description(skip_prompt=TRUE)
**Description**

Read, write and update the DESCRIPTION file. `read.description` reads the DESCRIPTION file in the current project directory and returns a named list. `write.description` writes the named list back to disk, overwriting the current DESCRIPTION file. Finally, `update_description` combines both functions by reading the DESCRIPTION file, updating or creating a field and writing the result back to disk.

**Usage**

```r
read.description()
write.description(description)
update_description(fieldname, value, after = NULL)
```

**Arguments**

- `description`: the DESCRIPTION file.
- `fieldname`: the name of the field.
- `value`: the new value.
- `after`: if the field name is new, the name of the field after which the element is placed.

**Details**

The 'Depends', 'Imports' and 'Suggests' fields are sorted before writing the DESCRIPTION file.

**Examples**

```r
## Not run:
description = read.description()
write.description(read.description())

#update date in description file
update_description("Date", format(Sys.Date(), "%Y-%m-%d"))

## End(Not run)
```
discretize_numbers

Discretize continuous numbers

Usage

discretize_numbers(x, min_size = 1, ...)

Arguments

x vector of numbers.
min_size minimum size of bins at the edges. Any bins smaller than this size are combined.
... Arguments passed on to get_breaks

N step size. The eventual intervals will be multiples of the divisors of N or multiples of N when multiples_only is TRUE. Defaults to 10.
max_breaks maximum amount of breaks, defaults to 10.
int_only whether only integer divisors of N may be used as breaks, defaults to TRUE.
multiples_only whether only multiples of N can be used as breaks, defaults to FALSE.

Details

The function get_breaks is called to create the boundaries between groups. It is called on default with limits = range(x) and with include_bounds = FALSE. This behaviour may be overridden with the ... argument, although it is advised not to do so to avoid empty groups.

NA values are preserved in the result.

Value

A factor with the same length as x, with labels indicating bins.

Examples

ages = round(rnorm(1000,50,10)); ages[1] = NA
discretize_numbers(ages)
### format_duration

**Format time duration**

**Description**

Format time duration

**Usage**

```
format_duration(start, end = Sys.time())
```

**Arguments**

- `start, end` date-time objects as obtained via `Sys.time`

**Value**

A string representation of the duration.

---

### frmt

**Format variable value**

**Description**

Creates a nice string representation of a variable value.

**Usage**

```
frmt(x, show_class = FALSE, use_quotes = TRUE)
```

**Arguments**

- `x` variable for which a string representation is created.
- `show_class` whether to show the class of `x`. Defaults to `FALSE`.
- `use_quotes` whether to use single quotation marks (default: `TRUE`).

**Value**

A character vector with the string representation of `x`.

**Examples**

```
frmt(c(1,2,3))
```
generic_implementations

Retrieve generic function implementations

Description

Obtains a list of classes for which the supplied generic function has an implementation.

Usage

generic_implementations(generic, remove_default = TRUE)

Arguments

generic       name of the generic function.
remove_default whether to keep the default generic implementation in the result.

Value

A vector with class names for which argument 'generic' has an implementation.

Note

Removes the default generic implementation

See Also

Other developer functions: crossref_description, load_packages, update_settings, valid_pkgname

Examples

# get a list of classes which have an implementation for graphics::plot
impls = generic_implementations('plot')

get_breaks

Create nice axis breaks for plots

Description

Set the breaks for a graph in nice positions.

Usage

gget_breaks(limits, N = 10, max_breaks = 10, int_only = TRUE,
multiples_only = FALSE, include_bounds = TRUE)

# ggplot_breaks(...)
Arguments

limits axis limits. May be either a vector of 2 elements with lower and upper bounds, or a single number (which is the upper bound, the lower bound is then assumed to be 0).

N step size. The eventual intervals will be multiples of the divisors of N or multiples of N when multiples_only is TRUE. Defaults to 10.

max_breaks maximum amount of breaks, defaults to 10.

int_only whether only integer divisors of N may be used as breaks, defaults to TRUE.

multiples_only whether only multiples of N can be used as breaks, defaults to FALSE.

include_bounds whether the resulting breaks should encompass min and max. Defaults to TRUE.

Details

get_breaks is the base function and creates a vector of breaks ggplot_breaks is a wrapper and makes usage easier in ggplot2. The limits of the axis may not be known beforehand, but ggplot_breaks receives it from ggplot and then creates nice breaks.

Value

A sorted numerical vector with breaks of length $|\text{max_breaks}|+2$ when include_bounds is TRUE and of size $|\text{max_breaks}|$ otherwise.

Examples

get_breaks(24, N=12, max_breaks=15)

## Not run:
ggplot() + scale_x_continuous(breaks = ggplot_breaks(N=12, max_breaks=15))
## End(Not run)
get_square_grid  

Specifies a square grid which fits N objects.

Description

The resulting grid will be of size a*a or a*(a+1) where a is an integer. It will therefore always be a square or or have one row/column more than columns/rows.

Usage

get_square_grid(N, moreRows = TRUE)

Arguments

N  
number of objects.

moreRows  
whether there should be more rows than columns if the resulting grid is not square. Defaults to more rows (TRUE).

Value

A named list with elements rows and columns specifying the size of the optimal grid.

Examples

get_square_grid(5)

inclusion_flowchart  

Patient flowchart

Description

Creates a patient flowchart which visualizes exclusions and updates the dataset.

Usage

inclusion_flowchart(dataset, node_text = "%s eligable patients", stratum = NULL)

exclude_patients(flowchart, dataset, exclusion_criterium, reason = deparse(substitute(exclusion_criterium)), node_text = "%s eligable patients", excluded_text = "%s excluded")
load_packages

Arguments

- **dataset**: The dataset, must be a data.frame.
- **node_text**: The text of the starting node, must be a string which can be interpreted by sprintf.
- **stratum**: An optional stratum, must be variable in dataset.
- **flowchart**: The flowchart object.
- **exclusion_criterium**: A boolean statement which is used to select patients to be discarded from the dataset.
- **reason**: An optional string to specify why patients were excluded. Defaults to the exclusion criterium.
- **excluded_text**: The text of the exclusion node, must be a string which can be interpreted by sprintf.

Value

A flowchart (when creating the flowchart), or updated dataset (when excluding patients).

Note

When excluding patients, the flowchart is updated 'behind the scenes' and is not returned.

Examples

```r
## Not run:
dataset = survival::lung; dataset$sex = factor(dataset$sex, labels = c("male", "female"))
flowchart = inclusion_flowchart(dataset)
dataset = exclude_patients(flowchart, dataset, status==1)  # exclude all patients who did not die
dataset = exclude_patients(flowchart, dataset, time<100)  # exclude patients with a short follow-up
flowchart  # print diagram
## End(Not run)
```

load_packages

Load and install packages

Description

Utility function to load and optionally install packages if they are missing. When the function terminates, packages are installed (if necessary) and loaded. Upgradeable packages are shown.

Usage

```r
load_packages(..., install_packages = TRUE, force_install = FALSE,
             show_outdated_packages = FALSE, default_loading_method = FALSE,
             return_library_statements = FALSE)
```
load_packages

Arguments

... list of package names.
install_packages whether to install the selected packages.
force_install whether to install packages even if they are installed already.
show_outdated_packages whether to show a list of packages which are outdated.
default_loading_method load according to the default R method using only library()
return_library_statements makes this function only return a string containing library() statements which can be paste into an R script.

Details

load_packages optionally installs, upgrades and attaches packages to the work space for a list of specified packages.

Value

Returns invisibly a list with additional package information and results of installing/upgrading and loading.

See Also

install.packages for installation of new packages, update.packages for updating outdated packages, library for load and attaching packages.

Other developer functions: crossref_description, generic_implementations, update_settings, valid_pkgname

Examples

```r
## Not run:
# Package names given one-by-one or in a vector
load_packages(c('magrittr', 'dplyr'))
load_packages('magrittr', 'dplyr')

# Package names may be unquoted
load_packages(magrittr, dplyr)
load_packages('magrittr', 'dplyr', install_packages=FALSE)

## End(Not run)
```
load_package_collection

*List package collections*

**Description**

List package collections

**Usage**

```r
load_package_collection(collection_name = names(list_package_collections()), ...)
```

```r
list_package_collections()
```

```r
list_common_packages()
```

```r
load_common_packages(...)
```

**Arguments**

- `collection_name`
  - One or multiple collection names. Must be in "data_import", "image_import", "ggplot", "grid", "survival", "processing", "shiny", "development".
- `...`
  - list of package names.

---

print.patient_flowchart

*Print the patient inclusion flowchart*

**Description**

Print the patient inclusion flowchart

**Usage**

```r
## S3 method for class 'patient_flowchart'
print(x, length = 7, ...)
```

**Arguments**

- `x`
  - an object used to select a method.
- `length`
  - Length of the arrows (to the right)
- `...`
  - further arguments passed to or from other methods.
**print.percentage_table**

*Print a formatted percentage table*

---

**Description**

Print a formatted percentage table

**Usage**

```r
## S3 method for class 'percentage_table'
print(x, ...)
```

**Arguments**

- **x**: An object of class `percentage_table`
- **...**: Further arguments passed to or from other methods.

**Examples**

```r
print(percentage_table(iris$Species))
```

---

**progressbar**

*Creates an animated progress bar*

---

**Description**

Creates an animated progress bar

**Usage**

```r
progressbar(format = "\[[\|/-\\]\] ", width = 20, refresh = 200, n_iterations = NULL)
render(object, ...)
```

```r
## S3 method for class 'fraction_progressbar'
render(object, progress, 
show_progress = c("nothing", "percentage"), ...)
```

```r
## S3 method for class 'iteration_progressbar'
render(object, progress, 
show_progress = c("nothing", "percentage", "iteration"), ...)
```

```r
## S3 method for class 'progressbar'
render(object, show_progress = c("nothing", "percentage", "iteration"), ...)
```
Arguments

- **format**: character vector containing the format of the animation. See ‘details’ for more information.
- **width**: progress bar width.
- **refresh**: refresh rate in milliseconds of the animation.
- **n_iterations**: optional parameter, specifies the number of total iterations. When updating the progress bar it is then sufficient to specify the current iteration number.
- **object**: animated progress bar.
- **progress**: either the iteration number (if n_iterations is set), or the progress fraction (in [0,1]).
- **show_progress**: how to show the progress. Either not to show it (default), show a percentage or if n_iterations is set to show the number of iterations.

Details

The format of the progress bar is given by a character vector. It consists of 5 parts:

1. the left border of the progress bar consisting of 0 or more characters.
2. a pair of square brackets containing a single character which represents the loaded area.
3. a pair of square brackets containing 0 or more characters. These are animated on the border between the loaded and unloaded area.
4. a pair of square brackets containing a single character which represents the unloaded area.
5. the right border of the progress bar consisting of 0 or more characters.

The format follows the following regular expression: `^.*?[.?][][.*?][.?]\.$`

Examples

```r
## Not run:
# simple progressbar
bar = progressbar(format = "[[][/-\][ ]]")
# fancy progressbar using UTF-8 codes
n_operations = 1000
bar2 = progressbar(format="\u25ba[\u2589][\u2580\u2584][\u3000]\u25c4", n_iterations=n_operations)

for(i in 1:n_operations) {
    cat("\r", render(bar),sep="")
    Sys.sleep(0.01)
}
## End(Not run)
```
**redundant_packages**

*Find redundant packages*

**Description**

Find redundant packages

**Usage**

`redundant_packages(packages)`

**Arguments**

- `packages`: list of package names.

**Details**

Certain packages have a direct dependency on other packages. In that case it is unnecessary to attach the latter packages. This function finds those packages and returns them in a named list. For each named item, the name is imported by the value in the list.

**Value**

A named list of packages names, where each value is a vector of packages already loading the corresponding package.

**Examples**

```r
## Not run:
#grid does not have be loaded since gridGraphics already does so.
redundant_packages(c("gridGraphics","grid"))

## End(Not run)
```

---

**rm_empty_rows**

*Remove empty rows*

**Description**

Remove empty rows

**Usage**

`rm_empty_rows(dataframe)`
Arguments
dataframe data.frame object.

Value
A data.frame with rows removed that only contain NA.

See Also
Other NA functions: rm_na

Examples
data <- rbind(c(1,2,3), c(1, NA, 4), c(4,6,7), c(NA, NA, NA), c(4, 8, NA))
rm_empty_rows(data)

---

rm_na Remove NA

Description
Remove NA

Usage
rm_na(x)

Arguments
x vector containing possible NA values.

Value
Vector without NA

See Also
Other NA functions: rm_empty_rows

Examples
rm_na(c(1,2,NA,54))
**rnd_dbl**

*Round number*

Description

Rounds a number to a specified amount of digits and returns the string value.

Usage

```r
rnd_dbl(dbl, digits = 3)
```

Arguments

- `dbl`: number to be rounded.
- `digits`: number of digits the number needs to be rounded to (defaults to 3).

Value

A string value of the number rounded to the specified amount of digits.

Examples

```r
rnd_dbl(1.26564, digits = 2)
```

**separate_values**

*Separate values*

Description

Separates real numbers from one another that are to close to each other. In the resulting set, the values are separated by a minimum distance, bounded by lower and upper limits and are constraint to be as close as possible to their original values.

Usage

```r
separate_values(X, distance = 0.05, min = 0, max = 1)
```

Arguments

- `X`: numerical vector of real numbers.
- `distance`: minimum distance between subsequent numbers. Must be a scalar or vector of size `|X|`.
- `min, max`: lower and upper limits.
Details

This function can be used for example to separate labels that are too close to one another. The resulting vector will create enough space, such that the labels do not overlap any more, yet are still close to their original values.

The output vector has the following properties. For all elements $e_i$, $\min \leq e_i \leq \max$. For the distance $D$ between $e_i$ and $e_{(i+1)}$, $D \geq \max(d_i, d_{(i+1)})$. And finally, the distance between $e_i$ and $X_i$ is minimized for all $e_i$.

Value

A numerical vector with the same length as $X$, with numbers bounded by min and max, close to their original values and with the minimum allowed distance between subsequent values.

Examples

separate_values(c(0.3, 0.4, 0.41), distance = 0.05, min = 0, max = 1)

---

sep_thousands

Add comma’s to separate thousands in numbers

Description

Add comma’s to separate thousands in numbers

Usage

sep_thousands(n)

Arguments

n a real number

Value

A string with the number and thousands separated by comma’s.

Examples

sep_thousands(13243.33) #13,243.33
Creates an animated spinner

Usage

```r
spinner(format = "|/-\", refresh = 200)
```

## S3 method for class 'spinner'
render(object, ...)

Arguments

- **format**: character vector containing the format of the animation. See 'details' for more information.
- **refresh**: refresh rate in milliseconds of the animation.
- **object**: animated spinner.
- **...**: further arguments passed to or from other methods.

Details

The format of the spinner simply consists of the characters in order which the spinner cycles through.

Examples

```r
## Not run:
sp = spinner("|/-\")
n_operations = 100

for(i in 1:n_operations) {
  cat("\r", render(sp),sep="")
  Sys.sleep(0.01)
}
## End(Not run)
```
startup
---

**Cleans R for use**

**Description**
Clears workspace, deletes all objects from global environment, clears graphics and (optionally) sets working directory.

**Usage**
```r
startup(removeObjects = TRUE, runGarbageCollection = TRUE,
        clearGraphics = TRUE, folder = NULL, verbose = TRUE)
```

**Arguments**
- `removeObjects` whether to remove objects from the workspace.
- `runGarbageCollection` whether to run the garbage collection.
- `clearGraphics` whether to clear the graphics from the R studio plots screen.
- `folder` folder name to set the current working directory.
- `verbose` whether to print informative messages during cleaning.

**Examples**
```r
## Not run: startup()
```

---

**stfu**

---

**S.T.F.U.: Stop Text From turning Up**

**Description**
S.T.F.U.: Stop Text From turning Up

**Usage**
```
stfu(expr)
```

**Arguments**
- `expr` expression to evaluate in silence.

**Value**
Returns invisibly the result of `expr`. 
**update_settings**

**Warning**

Make sure to call this function **always** directly on the expression and never indirectly e.g. via pipes. Example: `stfu(expr)` is correct, but `expr %>% stfu` will not hide the output. However, the `expr` argument itself may contain pipes.

**Examples**

```r
stfu(print("hi"))
```

---

**update_settings**  
*Update default function settings*

**Description**

Uses ellipsis parameter to update a list of default settings.

**Usage**

```r
update_settings(default, ...)
```

**Arguments**

- `default` named list of default values for settings.
- `...` optional settings values to override the default settings.

**Value**

The updated list of settings with updated values.

**See Also**

Other developer functions: `crossref_description`, `generic_implementations`, `load_packages`, `valid_pkgnames`

**Examples**

```r
foo = function(...) {
  default = list(a=1)
  settings = update_settings(default, ...)
}

## Not run: foo(a=2, b=3)
```
valid(pkg)
valid_funcname(func)

valid(pkg) # valid
valid(pkgname("hgutils")) # valid
valid(pkgname("ggplot2")) # valid
valid(pkgname("pkg2.-1")) # invalid

valid_funcname(.hgutils) # valid
valid_funcname("ggplot2") # valid
valid_funcname(".2pkg") # invalid
wrap_text_table

Wrap string table

Usage

wrap_text_table(string, exdent, min_size = 9, table_width = 80 - exdent)

Arguments

- **string**: character vector of strings to reformat.
- **exdent**: non-negative integer giving indentation of following lines in each paragraph
- **min_size**: minimal size where a table is constructed, otherwise elements are concatenated with ', '.
- **table_width**: table character width.

Value

A character vector of a wrapped table where rows are separated by the newline character.

See Also

- `str_wrap`, `get_square_grid`

Examples

```r
cat(wrap_text_table(LETTERS, exdent=0))
```
Index

.pkg_duplicated, 2
.regexl, 3

add_badges, 3
analyze_package_imports, 4
as.character.patient_flowchart, 5

create_contingency_table
 (create_table_one), 5
create_table_one, 5
create_text_table, 6
crossref_description, 7, 11, 15, 25, 26
description-functions, 8
discretize_numbers, 9

exclude_patients (inclusion_flowchart), 13

format_duration, 10
frmt, 10
generic_implementations, 7, 11, 15, 25, 26
get_breaks, 11
get_square_grid, 7, 13, 27
ggplot_breaks (get_breaks), 11

inclusion_flowchart, 13
install.packages, 15

library, 15
list_common_packages
 (load_package_collection), 16
list_package_collections
 (load_package_collection), 16
load_common_packages
 (load_package_collection), 16
load_package_collection, 16
load_packages, 7, 11, 14, 25, 26

make.names, 26

numeric_version, 7

percentage_table (create_table_one), 5
print.patient_flowchart, 16
print.percentage_table, 17
progressbar, 17

read.description
 (description-functions), 8
redundant_packages, 19
render (progressbar), 17
render.spinner (spinner), 23
rm_empty_rows, 19, 20
rm_na, 20, 20
rnd dbl, 21

sep_thousands, 22
separate_values, 21
spinner, 23
sprintf, 14
startup, 24
stfu, 24
str_wrap, 27
Sys.time, 10

update.packages, 15
update_description
 (description-functions), 8
update_settings, 7, 11, 15, 25, 26
valid_funcname (valid_pkgname), 26
valid_pkgname, 7, 11, 15, 25, 26

wrap_text_table, 27
write.description
 (description-functions), 8