Package 'hgutils'

June 2, 2025

Title Collection of Utility Functions

Version 0.2.19

Date 2025-06-02

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.

License GPL-3

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

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

Depends R (>= 4.4.0)

Imports crayon, dplyr, grDevices, lubridate, magrittr, methods, stats, stringr, utils

Suggests testthat

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.2

NeedsCompilation no

Author H.G. van den Boorn [aut, cre]

Maintainer H.G. van den Boorn <hvdboorn@gmail.com>

Repository CRAN

Date/Publication 2025-06-02 21:20:02 UTC

Contents

.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

.regexl *Extracts the matches from stringr::str_match[_all]*

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
)
```

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://www.travis-ci.com)
`show_code_coverage`	Whether to show the code coverage as a badge (see https://about.codecov.io/)

Examples

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

## End(Not run)
```

`analyze_package_imports`
Analyze package imports

Description

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

Usage

```
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]$")`

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

```
## 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

```
## 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.

assign_list *Assign variables in a list*

Description

Assign variables in a list

Usage

```
assign_list(x, envir = .GlobalEnv)
```

Arguments

x	A named list of values
envir	The environment in which the values are assigned, defaults to the global environment

Examples

```
assign_list(list(a=1, b=2))
```

create_table_one *Table one*

Description

Table one

Usage

```
create_table_one(df, numbers_as_categories = TRUE, deaths = NULL)

create_contigency_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	Input vector. Either a character vector, or something coercible to one.
table_width	table character width.
compact	whether to take only the necessary space (TRUE) or to fill out the table_width (FALSE).

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.
use_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)
```

---

default *Default value*

Description

Returns a default value for a scalar, to be used when the input is NA, NULL or has a length of 0.

Usage

```
default(x, default_value)
```

Arguments

x	A value
default_value	The replacement value for when x is NA, NULL or has a length of 0.

Value

The value of x when x is not NA, NULL or has a length of 0, and default_value otherwise.

Examples

```
default(NA, 0)
```

`description-functions` *Description functions*

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

```
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

```
## 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*

Description

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

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

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 `|max_breaks|+2` when `include_bounds` is `TRUE` and of size `|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"
)
```

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

```
## 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

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

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

```
## 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

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

list_package_collections()

list_common_packages()

load_common_packages(...)
```

Arguments

collection_name	One or multiple collection names. Must be in "data_import","image_import","ggplot","grid","su
...	list of package names.

print.patient_flowchart

Print the patient inclusion flowchart

Description

Print the patient inclusion flowchart

Usage

```
## 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

```
## 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

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

progressbar *Creates an animated progress bar*

Description

Creates an animated progress bar

Usage

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

render(object, ...)

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

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

## 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.
...	further arguments passed to or from other methods.
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

```
## 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)
```

---

progression_calculator

Creates a progression calculator which can display a loading bar and expected time to completion

---

Description

Creates a progression calculator which can display a loading bar and expected time to completion

Usage

```
progression_calculator(task_description, N)

## S3 method for class 'progression_calculator'
render(object, i, interval = 10, ...)
```

Arguments

task_description	A description of the task which is executed, if set to NA then no description if printed when using the render() function
N	The number of steps that are needed to complete the task
object	A progression calculator
i	The current iteration.
interval	The number of iterations to be completed before the progression calculator is updated.
...	further arguments passed to or from other methods.

Examples

```
## Not run:
#create progression calculator with 10 iterations
progress = progression_calculator("Example", N=10)
for(i in 1:10) {
  render(progress, i, interval=1) #render the calculator
  Sys.sleep(0.2)
}

## 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

```
## 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

```
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

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

`sep_thousands` *Adds comma's to separate thousands in numbers*

Description

Adds 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
```

`spinner` *Creates an animated spinner*

Description

Creates an animated spinner

Usage

```
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

```
## 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

```
startup(
  removeObjects = TRUE,
  runGarbageCollection = TRUE,
  clearGraphics = TRUE,
  folder = NULL,
  verbose = TRUE,
  seed = 37
)
```

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.
seed	the random set to be set with set.seed; is ignored if value is set to NULL.

Examples

```
## 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`.

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

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

translate_items *Translate item*

Description

Translate item

Usage

```
translate_items(vector, dict)
```

Arguments

vector	A vector whose values are to be translated.
dict	A named vector, whose names are keys in 'vector' to be replaced and whose values are the new values

Value

A vector with new values

Examples

```
v = c("A","B","C")
dict = c("A"="1")

translate_items(v, dict)
```

`update_settings` *Update default function settings*

Description

Uses ellipsis parameter to update a list of default settings.

Usage

```
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_pkgname()`

Examples

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

## Not run: foo(a=2, b=3)
```

valid_pkgname *Validate package and function names*

Description

Naming rule obtained from *'Writing R Extensions'* manual. The corresponding regular expression used for verifying the package name is `"[[:alpha:]][[:alnum:]\.]*[[:alnum:]]"`. For function names this is `"((?:[[:alpha:]]|\.(?![0-9]))[[:alnum:]_\.]*)"`

Usage

```
valid_pkgname(pkg)

valid_funcname(func)
```

Arguments

pkg	string vector containing package names. Can be a vector of strings with size of at least 1.
func	string vector containing function names. Can be a vector of strings with size of at least 1.

Value

A named logical indicating whether the package name is valid.

References

make.names, 'Writing R Extensions' manual.

See Also

Other developer functions: crossref_description(), generic_implementations(), load_packages(), update_settings()

Examples

```
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*

Description

Wrap string table

Usage

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

Arguments

`string`	Input vector. Either a character vector, or something coercible to one.
`exdent`	A non-negative integer giving the indent for all subsequent lines.
`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

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

Index