Package 'qdapRegex'

October 13, 2022

Type Package

Title Regular Expression Removal, Extraction, and Replacement Tools

Version 0.7.5

Maintainer Tyler Rinker <tyler.rinker@gmail.com>

Depends R (>= 3.5.0)

Imports stringi (>= 0.5-5)

Suggests testthat

LazyData TRUE

Description A collection of regular expression tools associated with the 'qdap' package that may be useful outside of the context of discourse analysis. Tools include removal/extraction/replacement of abbreviations, dates, dollar amounts, email addresses, hash tags, numbers, percentages, citations, person tags, phone numbers, times, and zip codes.

License GPL-2

URL https://github.com/trinker/qdapRegex

BugReports https://github.com/trinker/qdapRegex/issues

Collate 'S.R' 'bind.R' 'bind_or.R' 'c.extracted.R' 'case.R' 'cheat.R' 'utils.R' 'rm_default.R' 'escape.R' 'explain.R' 'grab.R' 'group.R' 'group_or.R' 'is.regex.R' 'pastex.R' 'print.extracted.R' 'print.regexr.R' 'qdapRegex-package.R' 'rm_.R' 'rm_abbreviation.R' 'rm_between.R' 'rm_bracket.R' 'rm_caps.R' 'rm_caps_phrase.R' 'rm_citation.R' 'rm_citation_tex.R' 'rm_city_state.R' 'rm_city_state_zip.R' 'rm_date.R' 'rm_dollar.R' 'rm_email.R' 'rm_emoticon.R' 'rm_endmark.R' 'rm_hash.R' 'rm_nchar_words.R' 'rm_non_ascii.R' 'rm_non_words.R' 'rm_number.R' 'rm_percent.R' 'rm_phone.R' 'rm_postal_code.R' 'rm_repeated_characters.R' 'rm_repeated_phrases.R' 'rm_repeated_words.R' 'rm_tag.R' 'rm_time.R' 'rm_title_name.R' 'rm_url.R' 'rm_white.R' 'rm_zip.R' 'validate.R'

Encoding UTF-8

RoxygenNote 7.1.2

NeedsCompilation no

Author Jason Gray [ctb],
Tyler Rinker [aut, cre]

Repository CRAN

Date/Publication 2022-05-02 07:10:02 UTC

R topics documented:

`bind` *Add Left/Right Character(s) Boundaries*

Description

This convenience function wraps left and right boundaries of each element of a character vector. The default is to use `"\b"` for left and right boundaries.

Usage

```
bind(
  ...,
  left = "\\b",
  right = left,
  dictionary = getOption("regex.library")
)
```

Arguments

`left`	A single length character vector to use as the left bound.
`right`	A single length character vector to use as the right bound.
`dictionary`	A dictionary of canned regular expressions to search within.
`...`	Regular expressions to add grouping parenthesis to a named expression from the default regular expression dictionary prefixed with single at (`@`) (e.g., `"@rm_hash"`) or a regular expression from `regex_supplement` dictionary prefixed with an at (`@`) (e.g., `"@time_12_hours"`).

Value

Returns a character vector.

See Also

paste0

Examples

```
bind(LETTERS, "[", "]")

## More useful default parameters/usage
x <- c("Computer is fun. Not too fun.", "No it's not, it's dumb.",
    "What should we do?", "You liar, it stinks!", "I am telling the truth!",
    "How can we be certain?", "There is no way.", "I distrust you.",
    "What are you talking about?", "Shall we move on?  Good then.",
    "I'm hungry.  Let's eat.  You already?")

Fry25 <- c("the", "of", "and", "a", "to", "in", "is", "you", "that", "it",
    "he", "was", "for", "on", "are", "as", "with", "his", "they",
    "I", "at", "be", "this", "have", "from")

gsub(pastex(list(bind(Fry25))), "[[ELIM]]", x)
```

---

bind_or *Boundary Wrap (Bind) and 'or' Concatenate Elements*

Description

A wrapper for bind and pastex that wraps each sub-expression element with left/right boundaries (\b by default) and then concatenate/joins bound strings with a regex 'or' ("|"). Equivalent to pastex(bind(...), sep = "|").

Usage

```
bind_or(..., group.all = TRUE, left = "\\b", right = left)
```

Arguments

group.all	logical. If TRUE the resulting 'or' concatenated elements will be wrapped with grouping parenthesis.
left	A single length character vector to use as the left bound.
right	A single length character vector to use as the right bound.
...	Regular expressions to paste together or a named expression from the default regular expression dictionary prefixed with single at (@) (e.g., "@rm_hash") or a regular expression from regex_supplement dictionary prefixed with an at (@) (e.g., "@time_12_hours").

Examples

```
bind_or(LETTERS)
bind_or("them", "those", "that", "these")
bind_or("them", "those", "that", "these", group.all = FALSE)
```

---

c.extracted *Combines a extracted Object*

Description

Combines a extracted object

Usage

```
## S3 method for class 'extracted'
c(x, ...)
```

Arguments

x	The extracted object
...	ignored

---

cheat *A Cheat Sheet of Common Regex Task Chunks*

Description

Print a cheat sheet of common regex task chunks. cheat prints a left justified version of regex_cheat.

Usage

```
cheat(dictionary = qdapRegex::regex_cheat, print = TRUE)
```

Arguments

dictionary	A dictionary of cheat terms. Default is regex_cheat.
print	logical. If TRUE the left justified output is printed to the console.

Value

Prints a cheat sheet of common regex tasks such as lookaheads. Invisibly returns regex_cheat.

See Also

regex_cheat

Examples

```
cheat()
```

---

escape *Escape Strings From Parsing*

Description

Escape literal beginning at (@) strings from **qdapRegex** parsing.

Usage

```
escape(pattern)
```

Arguments

pattern	A character string that should not be parsed.

Details

Many **qdapRegex** functions parse `pattern` strings beginning with an at character (@) and comparing against the default and supplemental (`regex_supplement`) dictionaries. This means that a string such as "@before_" will be returned as "\\w+?(?= ((%s|%s)\\b))". If the user wanted to use a regular expression that was literally "@before_" the `escape` function classes the character string and tells the **qdapRegex** functions not to parse it (i.e., keep it as a literal string).

Value

Returns a character vector of the class "escape" and "character".

Examples

```
escape("@rm_caps")

x <- "...character vector.  Default, \\code{@rm_caps} uses..."

rm_default(x, pattern = "@rm_caps")
rm_default(x, pattern = escape("@rm_caps"))
```

explain *Visualize Regular Expressions*

Description

Visualize regular expressions using https://regexper.com/

Usage

```
explain(
  pattern,
  open = FALSE,
  print = TRUE,
  dictionary = getOption("regex.library")
)
```

Arguments

pattern	A character string containing a regular expression or a character string starting with "@" that is a regular expression from a **qdapRegex** dictionary.
open	logical. If TRUE the default browser will attempt to open https://regexper.com/ page. Setting open = 2 will utilize an unstable visualization via https://www.debuggex.com. This approach utilizes a non-api scrape that is subject to change and not guaranteed to be stable. The regex is set to Python flavor which handles lookbehinds that the Java based https://regexper.com/ does not. This functionality was developed by Matthew Flickinger (see https://stackoverflow.com/a/27574103/1000343 for details). Note that the user must have **httr** installed or will be prompted if the package cannot be required.
print	logical. Should explain print output to the console?
dictionary	A dictionary of canned regular expressions to search within.

Details

Note that https://regexper.com/ is a Java based regular expression viewer. Lookbehind and negative lookbehinds are not respected.

Value

Prints https://regexper.com/ to the console, attempts to open the url to the visual representation provided by https://regexper.com/, and invisibly returns a list with the URLs.

Author(s)

Ananda Mahto, Matthew Flickinger, and Tyler Rinker <tyler.rinker@gmail.com>.

References

https://stackoverflow.com/a/27489977/1000343
https://regexper.com/
https://stackoverflow.com/a/27574103/1000343

See Also

https://regexper.com/

Examples

```
explain("\\s*foo[A-Z]\\d{2,3}")
explain("@rm_time")
## Not run:
explain("\\s*foo[A-Z]\\d{2,3}", open = TRUE)
explain("@rm_time", open = TRUE)

## End(Not run)
```

---

grab *Grab Regular Expressions from Dictionaries*

Description

convenience function to

Usage

```
grab(pattern, dictionary = getOption("regex.library"))
```

Arguments

pattern	A character string starting with "@" that is a regular expression from a **qdapRegex** dictionary.
dictionary	A dictionary of canned regular expressions to search within.

Details

Many R regular expressions contain doubled backslashes that are not used in other regex interpreters. Using cat can remove backslash escapes (see **Examples**) or URLencode if using in a url.

Value

Returns a single string regular expression from one of the **qdapRegex** dictionaries.

Examples

```
grab("@rm_white")
## Not run:
## Throws an error
grab("@foo")

## End(Not run)
cat(grab("@pages2"))
## Not run:
cat(grab("@pages2"), file="clipboard")

## End(Not run)
```

---

group *Group Regular Expressions*

Description

group - A wrapper for paste(collapse="|") that also searches the default and supplemental (regex_supplement) dictionaries for regular expressions before pasting them together with a pipe (|) separator.

Usage

```
group(..., left = "(", right = ")", dictionary = getOption("regex.library"))
```

Arguments

left	A single length character vector to use as the left bound.
right	A single length character vector to use as the right bound.
dictionary	A dictionary of canned regular expressions to search within.
...	Regular expressions to add grouping parenthesis to a named expression from the default regular expression dictionary prefixed with single at (@) (e.g., "@rm_hash") or a regular expression from regex_supplement dictionary prefixed with an at (@) (e.g., "@time_12_hours").

Value

Returns a single string of regular expressions with grouping parenthesis added.

Examples

```
group(LETTERS)
group(1)

(grouped <- group("(the|them)\\b", "@rm_zip"))
pastex(grouped)
```

group_or *Group Wrap and 'or' Concatenate Elements*

Description

A wrapper for group and pastex that wraps each sub-expression element with grouping parenthesis and then concatenate/joins grouped strings with a regex 'or' ("|"). Equivalent to pastex(group(...), sep = "|").

Usage

```
group_or(..., group.all = TRUE)
```

Arguments

group.all	logical. If TRUE the resulting 'or' concatenated elements will be wrapped with grouping parenthesis.
...	Regular expressions to paste together or a named expression from the default regular expression dictionary prefixed with single at (@) (e.g., "@rm_hash") or a regular expression from regex_supplement dictionary prefixed with an at (@) (e.g., "@time_12_hours").

Examples

```
group_or("@rm_hash", "@rm_tag")
group_or("them", "those", "that", "these")
group_or("them", "those", "that", "these", group.all = FALSE)
```

is.regex *Test Regular Expression Validity*

Description

Acts as a logical test of a regular expression's validity. is.regex uses gsub and tests for errors to determine a regular expression's validity. The regular expression must conform to R's regular expression rules (see ?regex for details about how R handles regular expressions).

Usage

```
is.regex(pattern)
```

Arguments

pattern	A regular expression to be tested.

Value

Returns a logical (TRUE is a valid regular expression).

See Also

gsub

Examples

```
is.regex("I|***")
is.regex("I|i")

sapply(regex_usa, is.regex)
sapply(regex_supplement, is.regex) ## `version` is not a valid regex
```

---

pastex *Paste Regular Expressions*

Description

pastex - A wrapper for paste(collapse="|") that also searches the default and supplemental (regex_supplement) dictionaries for regular expressions before pasting them together with a pipe (|) separator.

%|% - A binary operator version of pastex that joins two character strings with a regex or ("|"). Equivalent to pastex(x, y, sep="|").

%+% - A binary operator version of pastex that joins two character strings with no space. Equivalent to pastex(x, y, sep="").

Usage

```
pastex(..., sep = "|", dictionary = getOption("regex.library"))

x %|% y

x %+% y
```

Arguments

sep	The separator to use between the expressions when they are collapsed.
dictionary	A dictionary of canned regular expressions to search within.
x, y	Two regular expressions to paste together.
...	Regular expressions to paste together or a named expression from the default regular expression dictionary prefixed with single at (@) (e.g., "@rm_hash") or a regular expression from regex_supplement dictionary prefixed with an at (@) (e.g., "@time_12_hours").

Value

Returns a single string of regular expressions pasted together with pipe(s) (|).

Note

Note that while pastex is designed for pasting purposes it can also be used to call a single regex from the default regional dictionary or the supplemental dictionary (regex_supplement) (see **Examples**).

See Also

paste

Examples

```
x <- c("There is $5.50 for me.", "that's 45.6% of the pizza",
    "14% is $26 or $25.99", "It's 12:30 pm to 4:00 am")

pastex("@rm_percent", "@rm_dollar")
pastex("@rm_percent", "@time_12_hours")

rm_dollar(x, extract=TRUE, pattern=pastex("@rm_percent", "@rm_dollar"))
rm_dollar(x, extract=TRUE, pattern=pastex("@rm_dollar", "@rm_percent", "@time_12_hours"))

## retrieve regexes from dictionary
pastex("@rm_email")
pastex("@rm_url3")
pastex("@version")

## pipe operator (%|%)
"x" %|% "y"
"@rm_url" %|% "@rm_twitter_url"

## pipe operator (%p%)
"x" %+% "y"
"@rm_time" %+% "\\s[AP]M"

## Remove Twitter Short URL
x <- c("download file from http://example.com",
         "this is the link to my website http://example.com",
         "go to http://example.com from more info.",
         "Another url ftp://www.example.com",
         "And https://www.example.net",
         "twitter type: t.co/N1kq0F26tG",
         "still another one https://t.co/N1kq0F26tG :-)")

rm_twitter_url(x)
rm_twitter_url(x, extract=TRUE)

## Combine removing Twitter URLs and standard URLs
rm_twitter_n_url <- rm_(pattern="@rm_twitter_url" %|% "@rm_url")
rm_twitter_n_url(x)
```

```
rm_twitter_n_url(x, extract=TRUE)
```

`print.explain` *Prints a explain object*

Description

Prints a explain object

Usage

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

Arguments

`x`	The explain object
`...`	ignored

`print.extracted` *Prints a extracted Object*

Description

Prints a `extracted` object

Usage

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

Arguments

`x`	The `extracted` object.
`...`	Ignored.

`print.regexr` *Prints a regexr Object*

Description

Prints a `regexr` object

Usage

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

Arguments

`x`	The `regexr` object.
`...`	Ignored.

`qdapRegex` *qdapRegex: Regular Expression Removal, Extraction, & Replacement Tools for the* **qdap** *Package*

Description

qdapRegex is a collection of regular expression tools associated with the **qdap** package that may be useful outside of the context of discourse analysis. Tools include removal/extraction/replacement of abbreviations, dates, dollar amounts, email addresses, hash tags, numbers, percentages, citations, person tags, phone numbers, times, and zip codes.

Details

The **qdapRegex** package does not aim to compete with string manipulation packages such as **stringr** or **stringi** but is meant to provide access to canned, common regular expression patterns that can be used within **qdapRegex**, with **R**'s own regular expression functions, or add on string manipulation packages such as `stringr` and `stringi`.

regex_cheat *A dataset containing the regex chunk name, the regex string, and a description of what the chunk does.*

Description

A dataset containing the regex chunk name, the regex string, and a description of what the chunk does.

Usage

```
data(regex_cheat)
```

Format

A data frame with 6 rows and 3 variables

Details

- Name. The name of the regex chunk.
- Regex. The regex chunk.
- What it Does. Description of what the regex chunk does.

regex_supplement *Supplemental Canned Regular Expressions*

Description

A dataset containing a list of supplemental, canned regular expressions. The regular expressions in this data set are considered useful but have not been included in a formal function (of the type `rm_XXX`). Users can utilize the `rm_` function to generate functions that can sub/replace/extract as desired.

Usage

```
data(regex_supplement)
```

Format

A list with 24 elements

Details

The following canned regular expressions are included:

after_a single word after the word "a"

after_the single word after the word "the"

after_ find single word after ? word (? = user defined); note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own (user supplies (1) n before, (2) the point, & (3) n after)

around_ find n words (not including punctuation) before or after ? word (? = user defined); note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own (user supplies (1) n before, (2) the point, & (3) n after)

around2_ find n words (plus punctuation) before or after ? word (? = user defined); note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own

before_ find sing word before ? word (? = user defined); note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own

except_first find all occurrences of a substring except the first; regex pattern retrieved from StackOverflow's akrun: `https://stackoverflow.com/a/31458261/1000343`

hexadecimal substring beginning with hash (#) followed by either 3 or 6 select characters (a-f, A-F, and 0-9)

ip_address substring of four chunks of 1-3 consecutive digits separated with dots (.)

last_occurrence last occurrence of a delimiter; note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own (user supplies the delimiter)

pages substring with "pp." or "p.", optionally followed by a space, followed by 1 or more digits, optionally followed by a dash, optionally followed by 1 or more digits, optionally followed by a semicolon, optionally followed by a space, optionally followed by 1 or more digits; intended for extraction/removal purposes

pages2 substring 1 or more digits, optionally followed by a dash, optionally followed by 1 or more digits, optionally followed by a semicolon, optionally followed by a space, optionally followed by 1 or more digits; intended for validation purposes

punctuation punctuation characters (`[:punct:]`) with the ability to negate; note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own

run_split a regex that is useful for splitting strings in the characters runs (e.g., "wwxyyyzz" becomes "ww", "x", "yyy", "zz"); regex pattern retrieved from Robert Redd: `https://stackoverflow.com/a/29383435/1000343`

split_keep_delim regex string that splits on a delimiter and retains the delimiter

thousands_separator chunks digits > 4 into groups of 3 from right to left allowing for easy insertion of thousands separator; regex pattern retrieved from StackOverflow's stema: `https://stackoverflow.com/a/10612685/1000343`

time_12_hours substring of valid hours (1-12) followed by a colon (:) followed by valid minutes (0-60), followed by an optional space and the character chunk *am* or *pm*

version substring starting with "v" or "version" optionally followed by a space and then period separated digits for <major>.<minor>.<release>.<build>; the build sequence is optional and the "version"/"v" IS NOT contained in the substring

version2 substring starting with "v" or "version" optionally followed by a space and then period separated digits for <major>.<minor>.<release>.<build>; the build sequence is optional and the "version"/"v" IS contained in the substring

white_after_comma substring of white space after a comma

word_boundary A true word boundary that only includes alphabetic characters; based on https://www.rexegg.com/'s suggestion taken from discussion of true word boundaries; note contains "%s" that is replaced by sprintf and is not a valid regex on its own

word_boundary_left A true left word boundary that only includes alphabetic characters; based on https://www.rexegg.com/'s suggestion taken from discussion of true word boundaries

word_boundary_right A true right word boundary that only includes alphabetic characters; based on https://www.rexegg.com/'s suggestion taken from discussion of true word boundaries

youtube_id substring of the video id from a YouTube video; taken from Jacob Overgaard's submission found https://regex101.com/r/kU7bP8/1

Regexes from this data set can be added to the pattern argument of any rm_XXX function via an at sign (@) followed by a regex name from this data set (e.g., pattern = "@after_the") provided the regular expression does not contain non-regex such as sprintf character string %s.

Use qdapRegex:::examine_regex(regex_supplement) to interactively explore the regular expressions in regex_usa. This will provide a browser + console based break down of each regex in the dictionary.

Warning

Note that regexes containing %s are replaced by sprintf and are not a valid regex on their own. The S is useful for adding these missing %s parameters.

Examples

```
time <- rm_(pattern="@time_12_hours")
time("I will go at 12:35 pm")

x <- "v6.0.156 for Windows 2000/2003/XP/Vista
Server version 1.1.20
Client Manager version 1.1.24"

rm_default(x, pattern = "@version", extract=TRUE)
rm_default(x, pattern = "@version2", extract=TRUE)

x <- "this is 1000000 big 4356.  And little 123 number."
rm_default(x, pattern="@thousands_separator", replacement="\\1,")
rm_default(x, pattern="@thousands_separator", replacement="\\1.")

rm_default("I was,but it costs 10,000.", pattern="@white_after_comma",
    replacement=", ")

x <- "I like; the donuts; a lot"
strsplit(x, ";")
strsplit(x, S(grab("split_keep_delim"), ";"), perl=TRUE)
stringi::stri_split_regex(x, S(grab("split_keep_delim"), ";"))
```

```
stringi::stri_split_regex("I like; the donuts; a lot:cool",
    S(grab("split_keep_delim"), ";|:"))

## Grab words around a point
x <- c(
    "the magic word is e",
    "the dog is red and they are blue",
    "I am new but she is not new",
    "hello world",
    "why is it so cold?  Perhaps it is Winter.",
    "It is not true the 7 is 8.",
    "Is that my drink?"
)

rm_default(x, pattern = S("@around_", 1, "is", 1), extract=TRUE)
rm_default(x, pattern = S("@around_", 2, "is", 2), extract=TRUE)
rm_default(x, pattern = S("@around_", 1, "is|are|am", 1), extract=TRUE)
rm_default(x, pattern = S("@around_", 1, "is not|is|are|am", 1), extract=TRUE)
rm_default(x, pattern = S("@around_", 1,
    "is not|[Ii]s|[Aa]re|[Aa]m", 1), extract=TRUE)

x <- c(
    "hello world",
    "45",
    "45 & 5 makes 50",
    "x and y",
    "abc and def",
    "her him foo & bar for Jack and Jill then"
)

around_and <- rm_(pattern = S("@around_", 1, "and|\\&", 1), extract=TRUE)
around_and(x)

## Split runs into chunks
x <- "1111100000222000333300011110000111000"
strsplit(x, grab("@run_split"), per = TRUE)

## Not run:
library(qdap);library(ggplot2);library(reshape2)

out <- setNames(lapply(c("@after_a", "@after_the"), function(x) {
    o <- rm_default(stringi:::stri_trans_tolower(pres_debates2012$dialogue),
        pattern = x, extract=TRUE)
    m <- qdapTools::matrix2df(data.frame(freq=sort(table(unlist(o)), TRUE)), "word")
    m[m$freq> 7, ]
}), c("a", "the"))


dat <- setNames(Reduce(function(x, y) {
    merge(x, y, by = "word", all = TRUE)}, out), c("Word", "A", "THE"))

dat <- reshape2::melt(dat, id="Word", variable.name="Article", value.name="freq")
```

```
dat <- dat[order(dat$freq, dat$Word), ]

ord <- aggregate(freq ~ Word, dat, sum)

dat$word <- factor(dat$Word, levels=ord[order(ord[[2]]), 1])
ggplot(dat, aes(x=freq, y=Word)) + geom_point()+ facet_grid(~Article)

## End(Not run)

## remove/extract pages numbers
x <- c("I read p. 36 and then pp. 45-49", "it's on pp. 23-24;28")

rm_pages <- rm_(pattern="@pages", extract=TRUE)
rm_pages(x)

rm_default(x, pattern = "@pages")
rm_default(x, pattern = "@pages", extract=TRUE)
rm_default(x, pattern = "@pages2", extract=TRUE)

## Validate pages
page_val <- validate("@pages2", FALSE)
page_val(c(66, "78-82", "hello world", TRUE, "44-45; 56"))

## Split on last occurrence
x <- c(
    "test@aol@fg.mm.com",
    "test@hotmail.com",
    "test@xyz@rr@lk.edu",
    "test@abc.xx@zz.vv.net"
)

strsplit(x, S("@last_occurrence", "\\."), perl=TRUE)
strsplit(x, S("@last_occurrence", "@"), perl=TRUE)

## True Word Boundaries
x <- "this is _not a word666 and this is not a word too."
## Standard regex word boundary
rm_default(x, pattern=bind("not a word"))
## Alphabetic only word boundaries
rm_default(x, pattern=S("@word_boundary", "not a word"))

## Remove all but first occurrence of something
x <- c(
    "12-3=4-5=678-9",
    "ABC-D=EF2-GHI-JK3=L-MN=",
    "9-87=65",
    "a - de=4fgh --= i5jkl",
    NA
)

rm_default(x, pattern = S("@except_first", "-"))
rm_default(x, pattern = S("@except_first", "="))
```

regex_usa *Canned Regular Expressions (United States of America)*

Description

A dataset containing a list U.S. specific, canned regular expressions for use in various functions within the **qdapRegex** package.

Usage

```
data(regex_usa)
```

Format

A list with 54 elements

Details

The following canned regular expressions are included:

rm_abbreviation abbreviations containing single lower case or capital letter followed by a period and then an optional space (this must be repeated 2 or more times)

rm_between Remove characters between a left and right boundary including the boundaries; note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own

rm_between2 Remove characters between a left and right boundary NOT including the boundaries; note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own

rm_caps words containing 2 or more consecutive upper case letters and no lower case

rm_caps_phrase phrases of 1 word or more containing 1 or more consecutive upper case letters and no lower case; if phrase is one word long then phrase must be 2 or more consecutive capital letters

rm_citation substring that looks for in-text and parenthetical APA6 style citations (attempts to exclude references)

rm_citation2 substring that looks for in-text APA6 style citations (attempts to exclude references)

rm_citation3 substring that looks for parenthetical APA6 style citations (attempts to exclude references)

rm_city_state substring with *city* (single lower case word or multiple consecutive capitalized words before a comma and state) & *state* (2 consecutive capital letters)

rm_city_state_zip substring with *city* (single lower case word or multiple consecutive capitalized words before a comma and state) & *state* (2 consecutive capital letters) & *zip code* (exactly 5 or 5+4 consecutive digits)

rm_date dates in the form of 2 digit month, 2 digit day, and 2 or 4 digit year. Separator between month, day, and year may be dot (.), slash (/), or dash (-)

rm_date2 dates in the form of 3-9 letters followed by one or more spaces, 2 digits, a comma(,), one or more spaces, and 4 digits

rm_date3 dates in the form of XXXX-XX-XX; hyphen separated string of 4 digit year, 2 digit month, and 2 digit day

rm_date4 dates in the form of both `rm_date`, `rm_date2`, and `rm_date3`

rm_dollar substring with dollar sign ($) followed by (1) just dollars (no decimal), (2) dollars and cents (whole number and decimal), or (3) just cents (decimal value); dollars may contain commas

rm_email substring with (1) alphanumeric characters or dash (-), plus (+), or underscore (_) (*This may be repeated*) (2) followed by at (@), followed by the same regex sequence as before the at (@), and ending with dot (.) and 2-14 digits

rm_emoticon common emoticons (logic is complicated to explain in words) using ">?[:;=8XB]{1}[-~+o^]?[|\")(>DO>{pP3/]+|</?3|XD+|D:<|x[-~+o^]?[|\")(>DO>{pP3/]+" regex pattern; general pattern is optional hat character, followed by eyes character, followed by optional nose character, and ending with a mouth character

rm_endmark substring of the last endmark group in a string; endmarks include (! ? . * OR |)

rm_endmark3 substring of the last endmark group in a string; endmarks include (! ? OR .)

rm_endmark3 substring of the last endmark group in a string; endmarks include (! ? . * | ; OR :)

rm_hash substring that begins with a hash (#) followed by a word

rm_nchar_words substring of letters (that may contain apostrophes) n letters long (apostrophe not counted in length); note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own

rm_nchar_words2 substring of letters (that may contain apostrophes) n letters long (apostrophe counted in length); note contains `"%s"` that is replaced by `sprintf` and is not a valid regex on its own

rm_non_ascii substring of 2 digits or letters a-f inside of a left and right angle brace in the form of `"<a4>"`

rm_non_words substring of any character that isn't a letter, apostrophe, or single space

rm_number substring that may begin with dash (-) for negatives, and is (1) just whole number (no decimal), (2) whole number and decimal, or (3) just decimal value; regex pattern provided by Jason Gray

rm_percent substring beginning with (1) just whole number (no decimal), (2) whole number and decimal, or (3) just decimal value and followed by a percent sign (%)

rm_phone phone numbers in the form of optional country code, valid 3 digit prefix, and 7 digits (may contain hyphens and parenthesis); logic is complex to explain (see `https://stackoverflow.com/a/21008254/1000343` for more)

rm_postal_code U.S. state abbreviations (and District of Columbia) that is constrained to just possible U.S. state names, not just two consecutive capital letters; taken from Mike Hamilton's submission found `https://regexlib.com/REDetails.aspx?regexp_id=2177`

rm_repeated_characters substring with a repetition of repeated characters within a word; regex pattern retrieved from StackOverflow's, vks: `https://stackoverflow.com/a/29438461/1000343`

rm_repeated_phrases substring with a phrase (a sequence of 1 or more words) that is repeated 2 or more times (case is ignored; separating periods and commas are ignored); regex pattern retrieved from StackOverflow's, BrodieG: `https://stackoverflow.com/a/28786617/1000343`

rm_repeated_words substring with a word (marked with a boundary) that is repeat 2 or more times (case is ignored)

rm_tag substring that begins with an at (@) followed by a word

rm_tag2 Twitter substring that begins with an at (@) followed by a word composed of alpha-numeric characters and underscores, no longer than 15 characters

rm_title_name substring beginning with title (Mrs., Mr., Ms., Dr.) that is case independent or full title (Miss, Mizz, mizz) followed by a single lower case word or multiple capitalized words

rm_time substring that (1) must begin with 0-2 digits, (2) must be followed by a single colon (:), (3) optionally may be followed by either a colon (:) or a dot (.), (4) optionally may be followed by 1-infinite digits (if previous condition is true)

rm_time2 substring that is identical to `rm_time` with the additional search for Ante Meridiem/Post Meridiem abbreviations (e.g., AM, p.m., etc.)

rm_transcript_time substring that is specific to transcription time stamps in the form of HH:MM:SS.OS where OS is milliseconds. HH: and .OS are optional. The SS.OS period divide may also be a comma or additional colon. The HH:SS divid may also be a period. String may be affixed with pound sign (#).

rm_twitter_url Twitter short link/url; substring optionally beginning with *http*, followed by *t.co* ending on a space or end of string (whichever comes first)

rm_url substring beginning with *http*, *www.*, or *ftp* and ending on a space or end of string (whichever comes first); note that this regex is simple and may not cover all valid URLs or may include invalid URLs

rm_url2 substring beginning with *http*, *www.*, or *ftp* and more constrained than `rm_url`; based on @imme_emosol's response from `https://mathiasbynens.be/demo/url-regex`

rm_url3 substring beginning with *http* or *ftp* and more constrained than `rm_url` & `rm_url2` though light-weight, making it ideal for validation purposes; taken from @imme_emosol's response found `https://mathiasbynens.be/demo/url-regex`

rm_white substring of white space(s); this regular expression combines `rm_white_bracket`, `rm_white_colon`, `rm_white_comma`, `rm_white_endmark`, `rm_white_lead`, `rm_white_trail`, and `rm_white_multiple`

rm_white_bracket substring of white space(s) following left brackets ("{", "(", "[") or preceding right brackets ("}", ")", "]")

rm_white_colon substring of white space(s) preceding colon(s)/semicolon(s)

rm_white_comma substring of white space(s) preceding a comma

rm_white_endmark substring of white space(s) preceding a single occurrence/combination of period(s), question mark(s), and exclamation point(s)

rm_white_lead substring of leading white space(s)

rm_white_lead_trail substring of leading/trailing white space(s)

rm_white_multiple substring of multiple, consecutive white spaces

rm_white_punctuation substring of white space(s) preceding a comma or a single occurrence/combination of colon(s), semicolon(s), period(s), question mark(s), and exclamation point(s)

rm_white_trail substring of trailing white space(s)

rm_zip substring of 5 digits optionally followed by a dash and 4 more digits

Extra

Use `qdapRegex:::examine_regex()` to interactively explore the regular expressions in `regex_usa`. This will provide a browser + console based break down of each regex in the dictionary.

`rm_` *Remove/Replace/Extract Function Generator*

Description

Remove/replace/extract substrings from a string. A function generator used to make regex functions that operate typical of other **qdapRegex** `rm_XXX` functions. Use `rm_` for removal and `ex_` for extraction.

Usage

```
rm_(...)

ex_(...)
```

Arguments

`...`	Arguments passed to `rm_default`. Generally, `pattern` and `extract` are the most useful parameters to change. Arguments that can be set include: **text.var** The text variable. **trim** logical. If `TRUE` removes leading and trailing white spaces. **clean** logical. If `TRUE` extra white spaces and escaped character will be removed. **pattern** A character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. **replacement** Replacement for matched `pattern`. **extract** logical. If `TRUE` strings are extracted into a list of vectors. **dictionary** A dictionary of canned regular expressions to search within if `pattern` begins with `"@rm_"`. **...** Other arguments passed to `gsub`.

Value

Returns a function that operates typical of other **qdapRegex** `rm_XXX` functions but with user defined defaults.

See Also

`rm_default`

Examples

```
rm_digit <- rm_(pattern="[0-9]")
rm_digit("  I 12 li34ke ice56cream78.  ")

rm_lead <- rm_(pattern="^\\s+", trim = FALSE, clean = FALSE)
rm_lead("  I 12 li34ke ice56cream78.    ")

rm_all_except_letters <- rm_(pattern="[^ a-zA-Z]")
rm_all_except_letters("  I 12 li34ke ice56cream78.    ")

extract_consec_num <- rm_(pattern="[0-9]+", extract = TRUE)
extract_consec_num("  I 12 li34ke ice56cream78.  ")

## Using the supplemental dictionary dataset:
x <- "A man lives there!  The dog likes it. I want the map. I want an apple."

extract_word_after_the <- rm_(extract=TRUE, pattern="@after_the")
extract_word_after_a <- rm_(extract=TRUE, pattern="@after_a")
extract_word_after_the(x)
extract_word_after_a(x)

f <- rm_(pattern="@time_12_hours")
f("I will go at 12:35 pm")

x <- c(
    "test@aol.fg.com",
    "test@hotmail.com",
    "test@xyzrr.lk.edu",
    "test@abc.xx.zz.vv.net"
)

file_ext2 <- rm_(pattern="(?<=\\.)[a-z]*$", extract=TRUE)
tools::file_ext(x)
file_ext2(x)
```

`rm_abbreviation` *Remove/Replace/Extract Abbreviations*

Description

Remove/replace/extract abbreviations from a string containing lower case or capital letters followed by a period and then an optional space (this must be repeated 2 or more times).

Usage

```
rm_abbreviation(
  text.var,
  trim = !extract,
  clean = TRUE,
```

```
  pattern = "@rm_abbreviation",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_abbreviation(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_abbreviation",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_abbreviation uses the rm_abbreviation regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the abbreviations are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with abbreviations removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("I want $2.33 at 2:30 p.m. to go to A.n.p.",
    "She will send it A.S.A.P. (e.g. as soon as you can) said I.",
    "Hello world.", "In the U. S. A.")
rm_abbreviation(x)
ex_abbreviation(x)
```

`rm_between` *Remove/Replace/Extract Strings Between 2 Markers*

Description

Remove/replace/extract strings bounded between a left and right marker.

Usage

```
rm_between(
  text.var,
  left,
  right,
  fixed = TRUE,
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = FALSE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

rm_between_multiple(
  text.var,
  left,
  right,
  fixed = TRUE,
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = FALSE,
  include.markers = FALSE,
  merge = TRUE
)

ex_between(
  text.var,
  left,
  right,
```

```
  fixed = TRUE,
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = TRUE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

ex_between_multiple(
  text.var,
  left,
  right,
  fixed = TRUE,
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = TRUE,
  include.markers = FALSE,
  merge = TRUE
)
```

Arguments

text.var	The text variable.
left	A vector of character or numeric symbols as the left edge to extract.
right	A vector of character or numeric symbols as the right edge to extract.
fixed	logical. If TRUE regular expression special characters (c(".", "|", "(", ")", "[", "]", "{", "}", "^", "$", "*", "+", "?")) will be treated as typical characters. If the user wants to pass a regular expression with special characters then fixed = FALSE should be used.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the strings are extracted into a list of vectors.
include.markers	logical. If TRUE and extract = TRUE returns the markers (left/right) and the text between.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.
merge	logical. If TRUE the results of each bracket type will be merged by string. FALSE returns a named list of lists of vectors of markered text per marker type.

Value

Returns a character string with markers removed. If rm_between returns merged strings and is significantly faster. If rm_between_multiple the strings are optionally merged by left/right symbols. The latter approach is more flexible and names extracted strings by symbol boundaries, however, it is slower than rm_between.

See Also

gsub, rm_bracket, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <-  "I like [bots] (not)."

rm_between(x, "(", ")")
ex_between(x, "(", ")")
rm_between(x, c("(", "["), c(")", "]"))
ex_between(x, c("(", "["), c(")", "]"))

rm_between(x, c("(", "["), c(")", "]"), include.markers=FALSE)
ex_between(x, c("(", "["), c(")", "]"), include.markers=TRUE)

## multiple (naming and ability to keep separate bracket types but slower)
x <- c("Where is the /big dog#?",
    "I think he's @arunning@b with /little cat#.")

rm_between_multiple(x, "@a", "@b")
ex_between_multiple(x, "@a", "@b")
rm_between_multiple(x, c("/", "@a"), c("#", "@b"))
ex_between_multiple(x, c("/", "@a"), c("#", "@b"))

x2 <- c("Where is the L1big dogL2?",
    "I think he's 98running99 with L1little catL2.")
rm_between_multiple(x2, c("L1", 98), c("L2", 99))
ex_between_multiple(x2, c("L1", 98), c("L2", 99))

state <- c("Computer is fun. Not too fun.", "No it's not, it's dumb.",
    "What should we do?", "You liar, it stinks!", "I am telling the truth!",
    "How can we be certain?", "There is no way.", "I distrust you.",
    "What are you talking about?", "Shall we move on?  Good then.",
    "I'm hungry.  Let's eat.  You already?")

rm_between_multiple(state, c("is", "we"), c("too", "on"))

## Use Grouping
```

```
s <- "something before stuff $some text$ in between $1$ and after"
rm_between(s, "$", "$", replacement="<B>\\2<E>")

## Using regular expressions as boundaries (fixed =FALSE)
x <-  c(
    "There are 2.3 million species in the world",
    "There are 2.3 billion species in the world"
)

ex_between(x, left='There', right = '[mb]illion', fixed = FALSE, include=TRUE)
```

---

rm_bracket *Remove/Replace/Extract Brackets*

Description

Remove/replace/extract bracketed strings.

Usage

```
rm_bracket(
  text.var,
  pattern = "all",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = FALSE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

rm_round(
  text.var,
  pattern = "(",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = FALSE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

rm_square(
  text.var,
  pattern = "[",
  trim = TRUE,
```

```
  clean = TRUE,
  replacement = "",
  extract = FALSE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

rm_curly(
  text.var,
  pattern = "{",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = FALSE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

rm_angle(
  text.var,
  pattern = "<",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = FALSE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

rm_bracket_multiple(
  text.var,
  trim = TRUE,
  clean = TRUE,
  pattern = "all",
  replacement = "",
  extract = FALSE,
  include.markers = FALSE,
  merge = TRUE
)

ex_bracket(
  text.var,
  pattern = "all",
  trim = TRUE,
  clean = TRUE,
```

```
  replacement = "",
  extract = TRUE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

ex_bracket_multiple(
  text.var,
  trim = TRUE,
  clean = TRUE,
  pattern = "all",
  replacement = "",
  extract = TRUE,
  include.markers = FALSE,
  merge = TRUE
)

ex_angle(
  text.var,
  pattern = "<",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = TRUE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

ex_round(
  text.var,
  pattern = "(",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = TRUE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

ex_square(
  text.var,
  pattern = "[",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
```

```
  extract = TRUE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)

ex_curly(
  text.var,
  pattern = "{",
  trim = TRUE,
  clean = TRUE,
  replacement = "",
  extract = TRUE,
  include.markers = ifelse(extract, FALSE, TRUE),
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
pattern	The type of bracket (and encased text) to remove. This is one or more of the strings "curly"/"\{", "square"/"[", "round"/"(", "angle"/"<" and "all". These strings correspond to: {, [, (, < or all four types.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the bracketed text is extracted into a list of vectors.
include.markers	logical. If TRUE and extract = TRUE returns the markers (left/right) and the text between.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.
merge	logical. If TRUE the results of each bracket type will be merged by string. FALSE returns a named list of lists of vectors of bracketed text per bracket type.

Value

rm_bracket - returns a character string with multiple brackets removed. If extract = TRUE the results are optionally merged and named by bracket type. This is more flexible than rm_bracket but slower.

rm_round - returns a character string with round brackets removed.

rm_square - returns a character string with square brackets removed.

rm_curly - returns a character string with curly brackets removed.

`rm_angle` - returns a character string with angle brackets removed.

`rm_bracket_multiple` - returns a character string with multiple brackets removed. If `extract = TRUE` the results are optionally merged and named by bracket type. This is more flexible than `rm_bracket` but slower.

Author(s)

Martin Morgan and Tyler Rinker <tyler.rinker@gmail.com>.

References

https://stackoverflow.com/q/8621066/1000343

See Also

gsub, rm_between, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
examp <- structure(list(person = structure(c(1L, 2L, 1L, 3L),
    .Label = c("bob", "greg", "sue"), class = "factor"), text =
    c("I love chicken [unintelligible]!",
    "Me too! (laughter) It's so good.[interrupting]",
    "Yep it's awesome {reading}.", "Agreed. {is so much fun}")), .Names =
    c("person", "text"), row.names = c(NA, -4L), class = "data.frame")

examp
rm_bracket(examp$text, pattern = "square")
rm_bracket(examp$text, pattern = "curly")
rm_bracket(examp$text, pattern = c("square", "round"))
rm_bracket(examp$text)

ex_bracket(examp$text, pattern = "square")
ex_bracket(examp$text, pattern = "curly")
ex_bracket(examp$text, pattern = c("square", "round"))
ex_bracket(examp$text, pattern = c("square", "round"), merge = FALSE)
ex_bracket(examp$text)
ex_bracket(examp$tex, include.markers=TRUE)

## Not run:
library(qdap)
ex_bracket(examp$tex, pattern="curly") %>%
  unlist() %>%
  na.omit() %>%
  paste2()
```

```
## End(Not run)

x <- "I like [bots] (not). And <likely> many do not {he he}"

rm_round(x)
ex_round(x)
ex_round(x, include.marker = TRUE)

rm_square(x)
ex_square(x)

rm_curly(x)
ex_curly(x)

rm_angle(x)
ex_angle(x)

lapply(ex_between('She said, "I am!" and he responded..."Am what?".',
    left='"', right='"'), "[", c(TRUE, FALSE))
```

---

rm_caps *Remove/Replace/Extract All Caps*

Description

Remove/replace/extract 'all caps' words containing 2 or more consecutive upper case letters from a string.

Usage

```
rm_caps(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_caps",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_caps(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_caps",
  replacement = "",
  extract = TRUE,
```

```
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_caps uses the rm_caps regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the all caps strings are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with "all caps" removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("UGGG! When I use caps I am YELLING!")
rm_caps(x)
rm_caps(x, replacement="\\L\\1")
ex_caps(x)
```

`rm_caps_phrase` *Remove/Replace/Extract All Caps Phrases*

Description

Remove/replace/extract 'all caps' phrases containing 1 or more consecutive upper case letters from a string. If one word phrase the word must be 3+ letters long.

Usage

```
rm_caps_phrase(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_caps_phrase",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_caps_phrase(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_caps_phrase",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

`text.var`	The text variable.
`trim`	logical. If `TRUE` removes leading and trailing white spaces.
`clean`	trim logical. If `TRUE` extra white spaces and escaped character will be removed.
`pattern`	A character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. Default, `@rm_caps_phrae` uses the `rm_caps_phrase` regex from the regular expression dictionary from the `dictionary` argument.
`replacement`	Replacement for matched `pattern`.
`extract`	logical. If `TRUE` the all caps strings are extracted into a list of vectors.
`dictionary`	A dictionary of canned regular expressions to search within if `pattern` begins with `"@rm_"`.
`...`	Other arguments passed to `gsub`.

Value

Returns a character string with "all caps phrases" removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("UGGG! When I use caps I am YELLING!",
   "Or it may mean this is VERY IMPORTANT!",
   "or trying to make a LITTLE SEEM like IT ISN'T LITTLE"
)
rm_caps_phrase(x)
ex_caps_phrase(x)
```

---

rm_citation *Remove/Replace/Extract Citations*

Description

Remove/replace/extract APA6 style citations from a string.

Counts of normalized citations ("et al." to original author converted to author + year standarization).

Usage

```
rm_citation(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_citation",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_citation(
  text.var,
  trim = !extract,
  clean = TRUE,
```

```
  pattern = "@rm_citation",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

as_count(x, ...)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector (see **Details** for additional information). Default, @rm_citation uses the rm_citation regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the dates are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Ignored.
x	The output from ex_citation.

Details

The default regular expression used by rm_citation finds in-text and parenthetical citations. This behavior can be altered by using a secondary regular expression from the regex_usa data (or other dictionary) via (pattern = "@rm_citation2" or pattern = "@rm_citation3"). See **Examples** for example usage.

Value

Returns a character string with citations removed.

Returns a data.frame of Authors, Years, and n (counts).

Note

This function is experimental.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(),

rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
## All Citations
x <- c("Hello World (V. Raptor, 1986) bye",
    "Narcissism is not dead (Rinker, 2014)",
    "The R Core Team (2014) has many members.",
    paste("Bunn (2005) said, \"As for elegance, R is refined, tasteful, and",
        "beautiful. When I grow up, I want to marry R.\""),
    "It is wrong to blame ANY tool for our own shortcomings (Baer, 2005).",
    "Wickham's (in press) Tidy Data should be out soon.",
    "Rinker's (n.d.) dissertation not so much.",
    "I always consult xkcd comics for guidance (Foo, 2012; Bar, 2014).",
    "Uwe Ligges (2007) says, \"RAM is cheap and thinking hurts\""
)

rm_citation(x)
ex_citation(x)
as_count(ex_citation(x))
rm_citation(x, replacement="[CITATION HERE]")
## Not run:
qdapTools::vect2df(sort(table(unlist(rm_citation(x, extract=TRUE)))),
    "citation", "count")

## End(Not run)

## In-Text
ex_citation(x, pattern="@rm_citation2")

## Parenthetical
ex_citation(x, pattern="@rm_citation3")

## Not run:
## Mining Citation
if (!require("pacman")) install.packages("pacman")
pacman::p_load(qdap, qdapTools, dplyr, ggplot2)

url_dl("http://umlreading.weebly.com/uploads/2/5/2/5/25253346/whole_language_timeline-updated.docx")

parts <- read_docx("whole_language_timeline-updated.docx") %>%
    rm_non_ascii() %>%
    split_vector(split = "References", include = TRUE, regex=TRUE)

parts[[1]]

parts[[1]] %>%
    unbag() %>%
    ex_citation() %>%
```

```
    c()

## Counts
parts[[1]] %>%
    unbag() %>%
    ex_citation() %>%
    as_count()


## By line
ex_citation(parts[[1]])

## Frequency
cites <- parts[[1]] %>%
    unbag() %>%
    ex_citation() %>%
    c() %>%
    data_frame(citation=.) %>%
    count(citation) %>%
    arrange(n) %>%
    mutate(citation=factor(citation, levels=citation))

## Distribution of citations (find locations and then plot)
cite_locs <- do.call(rbind, lapply(cites[[1]], function(x){
    m <- gregexpr(x, unbag(parts[[1]]), fixed=TRUE)
    data.frame(
        citation=x,
        start = m[[1]] -5,
        end =  m[[1]] + 5 + attributes(m[[1]])[["match.length"]]
    )
}))

ggplot(cite_locs) +
    geom_segment(aes(x=start, xend=end, y=citation, yend=citation), size=3,
        color="yellow") +
    xlab("Duration") +
    scale_x_continuous(expand = c(0,0),
        limits = c(0, nchar(unbag(parts[[1]])) + 25)) +
    theme_grey() +
    theme(
        panel.grid.major=element_line(color="grey20"),
        panel.grid.minor=element_line(color="grey20"),
        plot.background = element_rect(fill="black"),
        panel.background = element_rect(fill="black"),
        panel.border = element_rect(colour = "grey50", fill=NA, size=1),
        axis.text=element_text(color="grey50"),
        axis.title=element_text(color="grey50")
    )

## End(Not run)
```

`rm_citation_tex` *Remove/Replace/Extract LaTeX Citations*

Description

Remove/replace/extract LaTeX citations from a string.

Usage

```
rm_citation_tex(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_citation_tex",
  replacement = "",
  extract = FALSE,
  split = extract,
  unlist.extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

ex_citation_tex(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_citation_tex",
  replacement = "",
  extract = TRUE,
  split = extract,
  unlist.extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

`text.var`	The text variable.
`trim`	logical. If `TRUE` removes leading and trailing white spaces.
`clean`	trim logical. If `TRUE` extra white spaces and escaped character will be removed.
`pattern`	A character string containing a regular expression (or character string).
`replacement`	Replacement for matched `pattern`.
`extract`	logical. If `TRUE` the dates are extracted into a list of vectors.
`split`	logical. If `TRUE` and `extract = TRUE` the bibkey will be removed from the LaTeX citation code curly braces and split on commas.

unlist.extract	logical. If TRUE the splits from between LaTeX citation code curly braces will be unlisted. if FALSE the list structure (1 per citation code curly brace) will be retained.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Additional arguments passed to rm_default.

Value

Returns a character string with citations (bibkeys) removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c(
    "I say \\parencite*{Ted2005, Moe1999} go there in \\textcite{Few2010} said to.",
    "But then \\authorcite{Ware2013} said it was so \\pcite[see][p. 22]{Get9999c}.",
    "then I \\citep[p. 22]{Foo1882c} him")

rm_citation_tex(x)
rm_citation_tex(x, replacement="[[CITATION]]")
ex_citation_tex(x)
```

---

rm_city_state *Remove/Replace/Extract City & State*

Description

Remove/replace/extract city (single lower case word or multiple consecutive capitalized words before a comma and state) & state (2 consecutive capital letters) from a string.

Usage

```
rm_city_state(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_city_state",
```

```
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_city_state(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_city_state",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_city_state uses the rm_city_state regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the city & state are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with city & state removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- paste0("I went to Washington Heights, NY for food! ",
   "It's in West ven,PA, near Bolly Bolly Bolly, CA!",
   "I like Movies, PG13")
rm_city_state(x)
ex_city_state(x)
```

`rm_city_state_zip` *Remove/Replace/Extract City, State, & Zip*

Description

Remove/replace/extract city (single lower case word or multiple consecutive capitalized words before a comma and state) + state (2 consecutive capital letters) + zip code (5 digits or 5 + 4 digits) from a string.

Usage

```
rm_city_state_zip(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_city_state_zip",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_city_state_zip(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_city_state_zip",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

`text.var`	The text variable.
`trim`	logical. If `TRUE` removes leading and trailing white spaces.
`clean`	trim logical. If `TRUE` extra white spaces and escaped character will be removed.

pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_city_state_zip uses the rm_city_state_zip regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the city, state, & zip are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with city, state, & zip removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- paste0("I went to Washington Heights, NY 54321 for food! ",
   "It's in West ven,PA 12345, near Bolly Bolly Bolly, CA12345-1234!",
   "hello world")
rm_city_state_zip(x)
ex_city_state_zip(x)
```

---

rm_date *Remove/Replace/Extract Dates*

Description

Remove/replace/extract dates from a string in the form of (1) XX/XX/XXXX, XX/XX/XX, XX-XX-XXXX, XX-XX-XX, XX.XX.XXXX, or XX.XX.XX OR (2) March XX, XXXX or Mar XX, XXXX OR (3) both forms.

Usage

```
rm_date(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_date",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_date(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_date",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector (see **Details** for additional information). Default, @rm_date uses the rm_date regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the dates are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Details

The default regular expression used by rm_date finds numeric representations not word/abbreviations. This means that "June 13, 2002" is not matched. This behavior can be altered (to include month names/abbreviations) by using a secondary regular expression from the regex_usa data (or other dictionary) via (pattern = "@rm_date2", pattern = "@rm_date3", or pattern = "@rm_date4"). See **Examples** for example usage.

Value

Returns a character string with dates removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
## Numeric Date Representation
x <- paste0("Format dates as 04/12/2014, 04-12-2014, 04.12.2014. or",
    " 04/12/14 but leaves mismatched: 12.12/2014")
rm_date(x)
ex_date(x)

## Word/Abbreviation Date Representation
x2 <- paste0("Format dates as Sept 09, 2002 or October 22, 1887",
  "but not 04-12-2014 and may match good 00, 9999")
rm_date(x2, pattern="@rm_date2")
ex_date(x2, pattern="@rm_date2")


## Year-Month-Day Representation
x3 <- sprintf("R uses time in this format %s.", Sys.time())
rm_date(x3, pattern="@rm_date3")

## Grab all types
ex_date(c(x, x2, x3), pattern="@rm_date4")
```

---

rm_default *Remove/Replace/Extract Template*

Description

Remove/replace/extract substring from a string. This is the template used by other **qdapRegex** rm_XXX functions.

Usage

```
rm_default(
  text.var,
  trim = !extract,
```

```
  clean = TRUE,
  pattern,
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_default(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern,
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the strings are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with substring removed.

See Also

rm_, gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
## Built in regex dictionary
rm_default("I live in Buffalo, NY 14217", pattern="@rm_city_state_zip")

## User defined regular expression
pat <- "(\\s*([A-Z][\\w-]*)+),\\s([A-Z]{2})\\s(?<!\\d)\\d{5}(?:[ -]\\d{4})?\\b"
rm_default("I live in Buffalo, NY 14217", pattern=pat)
```

`rm_dollar` *Remove/Replace/Extract Dollars*

Description

Remove/replace/extract dollars amounts from a string.

Usage

```
rm_dollar(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_dollar",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_dollar(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_dollar",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

`text.var`	The text variable.
`trim`	logical. If `TRUE` removes leading and trailing white spaces.
`clean`	trim logical. If `TRUE` extra white spaces and escaped character will be removed.

pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_dollar uses the rm_dollar regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the dollar strings are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with dollars removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <-  c("There is $5.50 for me.", "that's 45.6% of the pizza",
    "14% is $26 or $25.99", "Really?...$123,234.99 is not cheap.")

rm_dollar(x)
ex_dollar(x)
```

---

rm_email *Remove/Replace/Extract Email Addresses*

Description

Remove/replace/extract email addresses from a string.

Usage

```
rm_email(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_email",
```

```
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_email(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_email",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_email uses the rm_email regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the emails are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with email addresses removed.

Author(s)

Barry Rowlingson and Tyler Rinker <tyler.rinker@gmail.com>.

References

The email regular expression was taken from: https://stackoverflow.com/a/25077704/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- paste("fred is fred@foo.com and joe is joe@example.com - but @this is a
    twitter handle for twit@here.com or foo+bar@google.com/fred@foo.fnord")

x2 <- c("fred is fred@foo.com and joe is joe@example.com - but @this is a",
    "twitter handle for twit@here.com or foo+bar@google.com/fred@foo.fnord",
    "hello world")

rm_email(x)
rm_email(x, replacement = '<a href="mailto:\\1" target="_blank">\\1</a>')
ex_email(x)
ex_email(x2)
```

---

rm_emoticon *Remove/Replace/Extract Emoticons*

Description

Remove/replace/extract common emoticons from a string.

Usage

```
rm_emoticon(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_emoticon",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_emoticon(
  text.var,
  trim = !extract,
  clean = TRUE,
```

```
  pattern = "@rm_emoticon",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_emoticon uses the rm_emoticon regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the emoticons are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with emoticons removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("are :-)) it 8-D he XD on =-D they :D of :-) is :> for :o) that :-/",
  "as :-D I xD with :^) a =D to =) the 8D and :3 in =3 you 8) his B^D was")

rm_emoticon(x)
ex_emoticon(x)
```

`rm_endmark` *Remove/Replace/Extract Endmarks*

Description

Remove/replace/extract endmarks from a string.

Usage

```
rm_endmark(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_endmark",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_endmark(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_endmark",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

`text.var`	The text variable.
`trim`	logical. If `TRUE` removes leading and trailing white spaces.
`clean`	trim logical. If `TRUE` extra white spaces and escaped character will be removed.
`pattern`	A character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. Default, `@rm_endmark` uses the `rm_dollar` regex from the regular expression dictionary from the `dictionary` argument.
`replacement`	Replacement for matched `pattern`.
`extract`	logical. If `TRUE` the endmark strings are extracted into a list of vectors.
`dictionary`	A dictionary of canned regular expressions to search within if `pattern` begins with `"@rm_"`.
`...`	Other arguments passed to `gsub`.

Details

The default regular expression used by `rm_endmark` finds endmark punctuation used in the **qdap** package; this includes ! . ? * AND |. This behavior can be altered (to ; AND : or to use just ! . AND ?) by using a secondary regular expression from the `regex_usa` data (or other dictionary) via (`pattern = "@rm_endmark2"` or `pattern = "@rm_endmark3"`). See **Examples** for example usage.

Value

Returns a character string with endmarks removed.

See Also

`gsub`, `stri_extract_all_regex`

Other rm_ functions: `rm_abbreviation()`, `rm_between()`, `rm_bracket()`, `rm_caps_phrase()`, `rm_caps()`, `rm_citation_tex()`, `rm_citation()`, `rm_city_state_zip()`, `rm_city_state()`, `rm_date()`, `rm_default()`, `rm_dollar()`, `rm_email()`, `rm_emoticon()`, `rm_hash()`, `rm_nchar_words()`, `rm_non_ascii()`, `rm_non_words()`, `rm_number()`, `rm_percent()`, `rm_phone()`, `rm_postal_code()`, `rm_repeated_characters()`, `rm_repeated_phrases()`, `rm_repeated_words()`, `rm_tag()`, `rm_time()`, `rm_title_name()`, `rm_url()`, `rm_white()`, `rm_zip()`

Examples

```
x <- c("I like the dog.", "I want it *|", "I;",
    "Who is| that?", "Hello world", "You...")

rm_endmark(x)
ex_endmark(x)

rm_endmark(x, pattern="@rm_endmark2")
ex_endmark(x, pattern="@rm_endmark2")

rm_endmark(x, pattern="@rm_endmark3")
ex_endmark(x, pattern="@rm_endmark3")
```

---

rm_hash *Remove/Replace/Extract Hash Tags*

Description

Remove/replace/extract hash tags from a string.

Usage

```
rm_hash(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_hash",
```

```
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_hash(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_hash",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_hash uses the rm_hash regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the hash tags are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with hash tags removed.

Author(s)

stackoverflow's hwnd and Tyler Rinker <tyler.rinker@gmail.com>.

References

The hash tag regular expression was taken from: https://stackoverflow.com/a/25096474/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("@hadley I like #rstats for #ggplot2 work.",
   "Difference between #magrittr and #pipeR, both implement pipeline operators for #rstats:
      http://renkun.me/r/2014/07/26/difference-between-magrittr-and-pipeR.html @timelyportfolio",
   "Slides from great talk: @ramnath_vaidya: Interactive slides from Interactive Visualization
        presentation #user2014. http://ramnathv.github.io/user2014-rcharts/#1"
)

rm_hash(x)
rm_hash(rm_tag(x))
ex_hash(x)

## remove just the hash symbol
rm_hash(x, replace="\\3")
```

---

rm_nchar_words *Remove/Replace/Extract N Letter Words*

Description

Remove/replace/extract words that are n letters in length (apostrophes not counted).

Usage

```
rm_nchar_words(
  text.var,
  n,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_nchar_words",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_nchar_words(
```

```
  text.var,
  n,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_nchar_words",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
n	The number of letters counted in the word.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector (see **Details** for additional information). Default, @rm_nchar_words uses the rm_nchar_words regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the n letter words are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Details

The default regular expression used by rm_nchar_words counts letter length, not characters. This means that apostrophes are not include in the character count. This behavior can be altered (to include apostrophes in the character count) by using a secondary regular expression from the regex_usa data (or other dictionary) via (pattern = "@rm_nchar_words2"). See **Examples** for example usage.

Value

Returns a character string with n letter words removed.

Author(s)

stackoverflow's CharlieB and Tyler Rinker <tyler.rinker@gmail.com>.

References

The n letter/character word regular expression was taken from: https://stackoverflow.com/a/25243885/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- "This is Jon's dogs' 'bout there in a word Mike's re'y."
rm_nchar_words(x, 4)
ex_nchar_words(x, 4)

## Count characters (apostrophes and letters)
ex_nchar_words(x, 5, pattern = "@rm_nchar_words2")

## nchar range
rm_nchar_words(x, "1,2")

## Not run:
## Larger example
library(qdap)
ex_nchar_words(hamlet[["dialogue"]], 5)

## End(Not run)
```

---

rm_non_ascii *Remove/Replace/Extract Non-ASCII*

Description

Remove/replace/extract non-ASCII substring from a string. This is the template used by other **qdapRegex** rm_XXX functions.

Usage

```
rm_non_ascii(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_non_ascii",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ascii.out = TRUE,
  ...
```

```
)

ex_non_ascii(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_non_ascii",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ascii.out = TRUE,
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_non_ascii uses the rm_non_ascii regex from the regular expression dictionary from the dictionary argument. If extract = FALSE gsub is not used as with other rm_XXX functions, rather iconv with the sub argument set is used to conduct the subbing.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the all non-ASCII strings are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
ascii.out	logical. If TRUE output is given in non-ASCII format, otherwise "byte" is used.
...	ignored.

Value

Returns a character string with "all non-ascii" removed.

Warning

iconv is used within rm_non_ascii. iconv's behavior across operating systems may not be consistent.

Author(s)

stackoverflow's MrFlick, hwnd, and Tyler Rinker <tyler.rinker@gmail.com>.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("Hello World", "Ekstr\xf8m", "J\xf6reskog", "bi\xdfchen Z\xfcrcher")
Encoding(x) <- "latin1"
x

rm_non_ascii(x)
rm_non_ascii(x, replacement="<<FLAG>>")
ex_non_ascii(x)
ex_non_ascii(x, ascii.out=FALSE)

## simple regex to remove non-ascii
rm_default(x, pattern="[^ -~]")
ex_default(x, pattern="[^ -~]")
```

---

rm_non_words *Remove/Replace/Extract Non-Words*

Description

rm_non_words - Remove/replace/extract non-words (Anything that's not a letter or apostrophe; also removes multiple white spaces) from a string.

Usage

```
rm_non_words(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_non_words",
  replacement = " ",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_non_words(
  text.var,
```

```
  trim = !extract,
  clean = TRUE,
  pattern = "[^A-Za-z' ]+",
  replacement = " ",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_non_words uses the rm_non_words regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern (***Note:*** default is " ", whereas most **qdapRegex** functions replace with "").
extract	logical. If TRUE the non-words are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with non-words removed.

Note

Setting the argument extract = TRUE is not very useful. Use the following setup instead (see **Examples** for a demonstration).

```
rm_default(x, pattern = "[^A-Za-z' ]", extract=TRUE)
```

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c(
    "I like 56 dogs!",
    "It's seventy-two feet from the px290.",
    NA,
    "What",
    "that1is2a3way4to5go6.",
    "What do you*% want?  For real%; I think you'll see.",
    "Oh some <html>code</html> to remove"
)

rm_non_words(x)
ex_non_words(x)
```

`rm_number` *Remove/Replace/Extract Numbers*

Description

`rm_number` - Remove/replace/extract number from a string (works on numbers with commas, decimals and negatives).

`as_numeric` - A wrapper for `as.numeric(gsub(",", "", x))`, which removes commas and converts a list of vectors of strings to numeric. If the string cannot be converted to numeric `NA` is returned.

`as_numeric2` - A convenience function for `as_numeric` that unlists and returns a vector rather than a list.

Usage

```
rm_number(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_number",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

as_numeric(x)

as_numeric2(x)

ex_number(
  text.var,
  trim = !extract,
```

```
  clean = TRUE,
  pattern = "@rm_number",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_number uses the rm_number regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the numbers are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.
x	a character vector to convert to a numeric vector.

Value

rm_number - Returns a character string with number removed.

as_numeric - Returns a list of vectors of numbers.

as_numeric2 - Returns an unlisted vector of numbers.

References

The number regular expression was created by Jason Gray.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("-2 is an integer.  -4.3 and 3.33 are not.",
    "123,456 is 0 alot -123456 more than -.2", "and 3456789123 fg for 345.",
    "fg 12,345 23 .44 or 18.", "don't remove this 444,44", "hello world -.q")

rm_number(x)
ex_number(x)

##Convert to numeric
as_numeric(ex_number(x))   # retain list
as_numeric2(ex_number(x))  # unlist
```

`rm_percent` *Remove/Replace/Extract Percentages*

Description

Remove/replace/extract percentages from a string.

Usage

```
rm_percent(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_percent",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_percent(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_percent",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

`text.var`	The text variable.
`trim`	logical. If `TRUE` removes leading and trailing white spaces.

clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_percent uses the rm_percent regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the percentages are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with percentages removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <-  c("There is $5.50 for me.", "that's 45.6% of the pizza",
    "14% is $26 or $25.99")

rm_percent(x)
ex_percent(x)
```

---

rm_phone *Remove/Replace/Extract Phone Numbers*

Description

Remove/replace/extract phone numbers from a string.

Usage

```
rm_phone(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_phone",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_phone(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_phone",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_phone uses the rm_phone regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the phone numbers are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with phone numbers removed.

Author(s)

stackoverflow's Marius and Tyler Rinker <tyler.rinker@gmail.com>.

References

The phone regular expression was taken from: https://stackoverflow.com/a/21008254/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c(" Mr. Bean bought 2 tickets 2-613-213-4567 or 5555555555 call either one",
  "43 Butter Rd, Brossard QC K0A 3P0 - 613 213 4567",
  "Please contact Mr. Bean (613)2134567",
  "1.575.555.5555 is his #1 number",
  "7164347566",
  "I like 1234567 dogs"
)

rm_phone(x)
ex_phone(x)
```

---

rm_postal_code *Remove/Replace/Extract Postal Codes*

Description

Remove/replace/extract postal codes.

Usage

```
rm_postal_code(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_postal_code",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_postal_code(
```

```
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_postal_code",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_postal_code uses the rm_postal_code regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the city & state are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with postal codes removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("Anchorage, AK", "New York City, NY", "Some Place, Another Place, LA")
rm_postal_code(x)
ex_postal_code(x)
```

rm_repeated_characters *Remove/Replace/Extract Words With Repeating Characters*

Description

Remove/replace/extract words with repeating characters. The word must contain characters, each repeating at east 2 times

Usage

```
rm_repeated_characters(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_repeated_characters",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_repeated_characters(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_repeated_characters",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_repeated_characters uses the rm_repeated_characters regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the words with repeating characters are extracted into a list of vectors.

dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with percentages removed.

Author(s)

stackoverflow's vks and Tyler Rinker <tyler.rinker@gmail.com>.

References

https://stackoverflow.com/a/29438461/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- "aaaahahahahaha that was a good joke peep and pepper and pepe"
rm_repeated_characters(x)
ex_repeated_characters(x)
```

---

rm_repeated_phrases *Remove/Replace/Extract Repeating Phrases*

Description

Remove/replace/extract repeating phrases from a string.

Usage

```
rm_repeated_phrases(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_repeated_phrases",
  replacement = "",
  extract = FALSE,
```

```
  dictionary = getOption("regex.library"),
  ...
)

ex_repeated_phrases(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_repeated_phrases",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_repeated_phrases uses the rm_repeated_phrases regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the repeated phrases are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with percentages removed.

Author(s)

stackoverflow's BrodieG and Tyler Rinker <tyler.rinker@gmail.com>.

References

https://stackoverflow.com/a/28786617/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(),

rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c(
    "this is a big is a Big deal",
    "I want want to see",
    "I want, want to see",
    "I want...want to see see see how",
    "I like it. It is cool",
    "this is a big is a Big deal for those of, those of you who are."
)

rm_repeated_phrases(x)
ex_repeated_phrases(x)
```

---

rm_repeated_words *Remove/Replace/Extract Repeating Words*

Description

Remove/replace/extract repeating words from a string.

Usage

```
rm_repeated_words(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_repeated_words",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_repeated_words(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_repeated_words",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_repeated_words uses the rm_repeated_words regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the repeated words are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with percentages removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c(
    "this is a big is a Big deal",
    "I want want to see",
    "I want, want to see",
    "I want...want to see see see how",
    "I like it. It is cool",
    "this is a big is a Big deal for those of, those of you who are."
)

rm_repeated_words(x)
ex_repeated_words(x)
```

rm_tag *Remove/Replace/Extract Person Tags*

Description

Remove/replace/extract person tags from a string.

Usage

```
rm_tag(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_tag",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_tag(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_tag",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_tag uses the rm_tag regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the person tags are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Details

The default regex pattern `"(?<![@\w])@([a-z0-9_]+)\b"` is more liberal and searches for the at (@) symbol followed by any word. This can be accessed via `pattern = "@rm_tag"`. Twitter user names are more constrained. A second regex (`"(?<![@\w])@([a-z0-9_]{1,15})\b"`) is provide that contains the latter word to substring that begins with an at (@) followed by a word composed of alpha-numeric characters and underscores, no longer than 15 characters. This can be accessed via `pattern = "@rm_tag2"` (see **Examples**).

Value

Returns a character string with person tags removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_time(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("@hadley I like #rstats for #ggplot2 work.",
   "Difference between #magrittr and #pipeR, both implement pipeline operators for #rstats:
      http://renkun.me/r/2014/07/26/difference-between-magrittr-and-pipeR.html @timelyportfolio",
   "Slides from great talk: @ramnath_vaidya: Interactive slides from Interactive Visualization
        presentation #user2014. http://ramnathv.github.io/user2014-rcharts/#1",
    "tyler.rinker@gamil.com is my email",
    "A non valid Twitter is @abcdefghijklmnopqrstuvwxyz"
)

rm_tag(x)
rm_tag(rm_hash(x))
ex_tag(x)

## more restrictive Twitter regex
ex_tag(x, pattern="@rm_tag2")

## Remove only the @ sign
rm_tag(x, replacement = "\\3")
rm_tag(x, replacement = "\\3", pattern="@rm_tag2")
```

rm_time *Remove/Replace/Extract Time*

Description

rm_time - Remove/replace/extract time from a string.

rm_transcript_time - Remove/replace/extract transcript specific time stamps from a string.

as_time - Convert a time stamp removed by rm_time or rm_transcript_time to a standard time format (HH:SS:MM.OS) and optionally convert to as.POSIXlt.

as_time - A convenience function for as_time that unlists and returns a vector rather than a list.

Usage

```
rm_time(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_time",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

rm_transcript_time(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_transcript_time",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

as_time(x, as.POSIXlt = FALSE, millisecond = TRUE)

as_time2(x, ...)

ex_time(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_time",
  replacement = "",
  extract = TRUE,
```

```
  dictionary = getOption("regex.library"),
  ...
)

ex_transcript_time(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_transcript_time",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector (see **Details** for additional information). Default, @rm_time uses the rm_time regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the times are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.
x	A list with extracted time stamps.
as.POSIXlt	logical. If TRUE the output will be converted to as.POSIXlt.
millisecond	logical. If TRUE milliseconds are retained. If FALSE they are rounded and added to seconds.

Details

The default regular expression used by rm_time finds time with no AM/PM. This behavior can be altered by using a secondary regular expression from the regex_usa data (or other dictionary) via (pattern = "@rm_time2". See **Examples** for example usage.

Value

Returns a character string with time removed.

Note

. . . in as_time2 are the other arguments passed to as_time.

Author(s)

stackoverflow's hwnd and Tyler Rinker <tyler.rinker@gmail.com>.

References

The time regular expression was taken from: https://stackoverflow.com/a/25111133/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_title_name(), rm_url(), rm_white(), rm_zip()

Examples

```
x <-  c("R uses 1:5 for 1, 2, 3, 4, 5.",
    "At 3:00 we'll meet up and leave by 4:30:20",
    "We'll meet at 6:33.", "He ran it in :22.34")

rm_time(x)
ex_time(x)

## With AM/PM
x <- c(
    "I'm getting 3:04 AM just fine, but...",
    "for 10:47 AM I'm getting 0:47 AM instead.",
    "no time here",
    "Some time has 12:04 with no AM/PM after it",
    "Some time has 12:04 a.m. or the form 1:22 pm"
)

ex_time(x)
ex_time(x, pat="@rm_time2")
rm_time(x, pat="@rm_time2")
ex_time(x, pat=pastex("@rm_time2", "@rm_time"))

# Convert to standard format
as_time(ex_time(x))
as_time(ex_time(x), as.POSIXlt = TRUE)
as_time(ex_time(x), as.POSIXlt = FALSE, millisecond = FALSE)
```

```
# Transcript specific time stamps
x2 <-c(
    '08:15 8 minutes and 15 seconds 00:08:15.0',
    '3:15 3 minutes and 15 seconds not 1:03:15.0',
    '01:22:30 1 hour 22 minutes and 30 seconds 01:22:30.0',
    '#00:09:33-5# 9 minutes and 33.5 seconds 00:09:33.5',
    '00:09.33,75 9 minutes and 33.5 seconds 00:09:33.75'
)

rm_transcript_time(x2)
(out <- ex_transcript_time(x2))

as_time(out)
as_time(out, TRUE)
as_time(out, millisecond = FALSE)

## Not run:
if (!require("pacman")) install.packages("pacman")
pacman::p_load(chron)
lapply(as_time(out), chron::times)
lapply(as_time(out, , FALSE), chron::times)

## End(Not run)
```

---

rm_title_name *Remove/Replace/Extract Title + Person Name*

Description

Remove/replace/extract title (honorific) + person name(s) from a string.

Usage

```
rm_title_name(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_title_name",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_title_name(
  text.var,
  trim = !extract,
  clean = TRUE,
```

```
  pattern = "@rm_title_name",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_title_name uses the rm_title_name regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the person tags are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with person tags removed.

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_url(), rm_white(), rm_zip()

Examples

```
x <- c("Dr. Brend is mizz hart's in mrs. Holtz's.",
    "Where is mr. Bob Jr. and Ms. John Kennedy?")

rm_title_name(x)
ex_title_name(x)
```

rm_url *Remove/Replace/Extract URLs*

Description

rm_url - Remove/replace/extract URLs from a string.

rm_twitter_url - Remove/replace/extract Twitter Short URLs from a string.

Usage

```
rm_url(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_url",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

rm_twitter_url(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_twitter_url",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_url(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_url",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

ex_twitter_url(
  text.var,
  trim = !extract,
```

```
  clean = TRUE,
  pattern = "@rm_twitter_url",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_url uses the rm_url regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the URLs are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Details

The default regex pattern "(http[^]*)|(www\.[^]*)" is more liberal. More constrained versions can be accessed via pattern = "@rm_url2" & pattern = "@rm_url3" see **Examples**).

Value

Returns a character string with URLs removed.

References

The more constrained url regular expressions ("@rm_url2" and "@rm_url3" was adapted from imme_emosol's response: https://mathiasbynens.be/demo/url-regex

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_white(), rm_zip()

Examples

```
x <- " I like www.talkstats.com and http://stackoverflow.com"
rm_url(x)
rm_url(x, replacement = '<a href="\\1" target="_blank">\\1</a>')
ex_url(x)

ex_url(x, pattern = "@rm_url2")
ex_url(x, pattern = "@rm_url3")

## Remove Twitter Short URL
x <- c("download file from http://example.com",
         "this is the link to my website http://example.com",
         "go to http://example.com from more info.",
         "Another url ftp://www.example.com",
         "And https://www.example.net",
         "twitter type: t.co/N1kq0F26tG",
         "still another one https://t.co/N1kq0F26tG :-)")

rm_twitter_url(x)
ex_twitter_url(x)

## Combine removing Twitter URLs and standard URLs
rm_twitter_n_url <- rm_(pattern=pastex("@rm_twitter_url", "@rm_url"))
rm_twitter_n_url(x)
rm_twitter_n_url(x, extract=TRUE)
```

---

rm_white *Remove/Replace/Extract White Space*

Description

`rm_white` - Remove multiple white space (> 1 becomes a single white space), white space before a comma, white space before a single or consecutive combination of a colon, semicolon, or endmark (period, question mark, or exclamation point), white space after a left bracket ("", "(", "[") or before a right bracket ("", ")", "]"), leading or trailing white space.

`rm_white_bracket` - Remove white space after a left bracket ("", "(", "[") or before a right bracket ("", ")", "]").

`rm_white_colon` - Remove white space before a single or consecutive combination of a colon, semicolon.

`rm_white_comma` - Remove white space before a comma.

`rm_white_endmark` - Remove white space before endmark(s) (".", "?", "!").

`rm_white_lead` - Remove leading white space.

`rm_white_lead_trail` - Remove leading or trailing white space.

`rm_white_trail` - Remove trailing white space.

`rm_white_multiple` - Remove multiple white space (> 1 becomes a single white space).

rm_white_punctuation - Remove multiple white space before a comma, white space before a single or consecutive combination of a colon, semicolon, or endmark (period, question mark, or exclamation point).

Usage

```
rm_white(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_bracket(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_bracket",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white_bracket(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_bracket",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

```
rm_white_colon(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_colon",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white_colon(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_colon",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_comma(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_comma",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white_comma(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_comma",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_endmark(
  text.var,
  trim = !extract,
```

```
  clean = TRUE,
  pattern = "@rm_white_endmark",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white_endmark(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_endmark",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_lead(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white_lead",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white_lead(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white_lead",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_lead_trail(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white_lead_trail",
  replacement = "",
  extract = FALSE,
```

```
  dictionary = getOption("regex.library"),
  ...
)

ex_white_lead_trail(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white_lead_trail",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_trail(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white_trail",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white_trail(
  text.var,
  trim = FALSE,
  clean = FALSE,
  pattern = "@rm_white_trail",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_multiple(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_multiple",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)
```

```
ex_white_multiple(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_multiple",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)

rm_white_punctuation(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_punctuation",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_white_punctuation(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_white_punctuation",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_dollar uses the rm_dollar regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the dollar strings are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with extra white space removed.

Author(s)

rm_white_endmark/rm_white_punctuation - stackoverflow's hwnd and Tyler Rinker <tyler.rinker@gmail.com>.

References

The rm_white_endmark/rm_white_punctuation regular expression was taken from: https://stackoverflow.com/a/25464921/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_zip()

Examples

```
x <-  c(" There is ( $5.50 ) for , me . ", " that's [ 45.6% ] of! the pizza !",
    "     14% is { $26  } or $25.99 ?", "Oh ;  here's colon : Yippee !")

rm_white(x)
rm_white_bracket(x)
rm_white_colon(x)
rm_white_comma(x)
rm_white_endmark(x)
rm_white_lead(x)
rm_white_trail(x)
rm_white_lead_trail(x)
rm_white_multiple(x)
rm_white_punctuation(x)
```

---

rm_zip *Remove/Replace/Extract Zip Codes*

Description

Remove/replace/extract zip codes from a string.

Usage

```
rm_zip(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_zip",
  replacement = "",
  extract = FALSE,
  dictionary = getOption("regex.library"),
  ...
)

ex_zip(
  text.var,
  trim = !extract,
  clean = TRUE,
  pattern = "@rm_zip",
  replacement = "",
  extract = TRUE,
  dictionary = getOption("regex.library"),
  ...
)
```

Arguments

text.var	The text variable.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Default, @rm_zip uses the rm_zip regex from the regular expression dictionary from the dictionary argument.
replacement	Replacement for matched pattern.
extract	logical. If TRUE the zip codes are extracted into a list of vectors.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".
...	Other arguments passed to gsub.

Value

Returns a character string with U.S. 5 and 5+4 zip codes removed.

Author(s)

stackoverflow's hwnd and Tyler Rinker <tyler.rinker@gmail.com>.

References

The time regular expression was taken from: https://stackoverflow.com/a/25223890/1000343

See Also

gsub, stri_extract_all_regex

Other rm_ functions: rm_abbreviation(), rm_between(), rm_bracket(), rm_caps_phrase(), rm_caps(), rm_citation_tex(), rm_citation(), rm_city_state_zip(), rm_city_state(), rm_date(), rm_default(), rm_dollar(), rm_email(), rm_emoticon(), rm_endmark(), rm_hash(), rm_nchar_words(), rm_non_ascii(), rm_non_words(), rm_number(), rm_percent(), rm_phone(), rm_postal_code(), rm_repeated_characters(), rm_repeated_phrases(), rm_repeated_words(), rm_tag(), rm_time(), rm_title_name(), rm_url(), rm_white()

Examples

```
x <- c("Mr. Bean bought 2 tickets 2-613-213-4567",
  "43 Butter Rd, Brossard QC K0A 3P0 - 613 213 4567",
  "Rat Race, XX, 12345",
  "Ignore phone numbers(613)2134567",
  "Grab zips with dashes 12345-6789 or no space before12345-6789",
  "Grab zips with spaces 12345 6789 or no space before12345 6789",
  "I like 1234567 dogs"
)

rm_zip(x)
ex_zip(x)

## ======================== ##
## BUILD YOUR OWN FUNCTION ##
## ======================== ##

## example from: https://stackoverflow.com/a/26092576/1000343
zips <- data.frame(id = seq(1, 6),
    address = c("Company, 18540 Main Ave., City, ST 12345",
    "Company 18540 Main Ave. City ST 12345-0000",
    "Company 18540 Main Ave. City State 12345",
    "Company, 18540 Main Ave., City, ST 12345 USA",
    "Company, One Main Ave Suite 18540m, City, ST 12345",
    "company 12345678")
)

## Function to grab even if a character follows the zip

# paste together a more flexible regular expression
pat <- pastex(
    "@rm_zip",
    "(?<!\\d)\\d{5}(?!\\d)",
    "(?<!\\d)\\d{5}-\\d{4}(?!\\d)"
)
# Create your own function that extract is set to TRUE
ex_zip2 <- rm_(pattern=pat, extract=TRUE)
```

```
ex_zip2(zips$address)

## Function to extract just 5 digit zips

ex_zip3 <- rm_(pattern="(?<!\\d)\\d{5}(?!\\d)", extract=TRUE)
ex_zip3(zips$address)
```

S *Use C-style String Formatting Commands*

Description

Convenience wrapper for sprintf that allows recycling of ... of length one.

Usage

```
S(x, ...)
```

Arguments

x	A single string containing "%s".
...	A vector of substitutions equal in length to the number of "%s" in x or of length one (if length one ... will be recycled).

Value

Returns a string with "%s" replaced.

See Also

sprintf

Examples

```
S("@after_", "the", "the")
# Recycle
S("@after_", "the")
S("@rm_between", "LEFT", "RIGHT")
```

TC *Upper/Lower/Title Case*

Description

TC - Capitalize titles according to traditional capitalization rules.

Usage

```
TC(text.var, lower = NULL, ...)

L(text.var, ...)

U(text.var, ...)
```

Arguments

text.var	The text variable.
lower	A vector of words to retain lower case for (unless first or last word).
...	Other arguments passed to: stri_trans_tolower, stri_trans_toupper, and stri_trans_totitle.

Details

Case wrapper functions for **stringi**'s stri_trans_tolower, stri_trans_toupper, and stri_trans_totitle. Functions are useful within **magrittr** style chaining.

Value

Returns a character vector with new case (lower, upper, or title).

Note

TC utilizes additional rules for capitalization beyond stri_trans_totitle that include:

1. Capitalize the first & last word
2. Lowercase articles, coordinating conjunctions, & prepositions
3. Lowercase "to" in an infinitive

See Also

stri_trans_tolower, stri_trans_toupper, stri_trans_totitle

Examples

```
y <- c(
   "I'm liking it but not too much.",
   "How much are you into it?",
   "I'd say it's yet awesome yet."
)
L(y)
U(y)
TC(y)
```

---

validate *Regex Validation Function Generator*

Description

Generate function to validate regular expressions.

Usage

```
validate(
  pattern,
  single = TRUE,
  trim = FALSE,
  clean = FALSE,
  dictionary = getOption("regex.library")
)
```

Arguments

pattern	A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector.
single	logical. If TRUE only returns true if the output string is of length one. If FALSE multiple strings and multiple outputs are accepted.
trim	logical. If TRUE removes leading and trailing white spaces.
clean	trim logical. If TRUE extra white spaces and escaped character will be removed.
dictionary	A dictionary of canned regular expressions to search within if pattern begins with "@rm_".

Value

Returns a function that operates typical of other **qdapRegex** rm_XXX functions but with user defined defaults.

Warning

validate uses **qdapRegex**'s built in regular expressions. As this patterns are used for text analysis they tend to be flexible and thus liberal. The user may wish to define more conservative validation regular expressions and supply to pattern.

Examples

```
## Single element email
valid_email <- validate("@rm_email")
valid_email(c("tyler.rinker@gmail.com", "@trinker"))

## Multiple elements
valid_email_1 <- validate("@rm_email", single=FALSE)
valid_email_1(c("tyler.rinker@gmail.com", "@trinker"))

## single element address
valid_address <- validate("@rm_city_state_zip")
valid_address("Buffalo, NY 14217")
valid_address("buffalo,NY14217")
valid_address("buffalo NY 14217")

valid_address2 <- validate(paste0("(\\b([A-Z][\\w-]*)+),",
    "\\s([A-Z]{2})\\s(?<!\\d)\\d{5}(?:[ -]\\d{4})?\\b"))
valid_address2("Buffalo, NY 14217")
valid_address2("buffalo, NY 14217")
valid_address2("buffalo,NY14217")
valid_address2("buffalo NY 14217")
```

Index