Package 'strex'

November 30, 2018

Title Extra String Manipulation Functions

Version 0.1.3

Description There are some things that I wish were easier with the 'stringr' or 'stringi' packages. The foremost of these is the extraction of numbers from strings. 'stringr' and 'stringi' make you figure out the regular expression for yourself; 'strex' takes care of this for you. There are many other handy functionalities in 'strex'. Contributions to this package are encouraged: it is intended as a miscellany of string manipulation functions that cannot be found in 'stringi' or 'stringr'.

Depends stringr

License GPL-3

Encoding UTF-8

LazyData true

LinkingTo Rcpp

Imports Rcpp, magrittr, checkmate, matrixStats, ore (>= 1.4.0), purrr, rlang, tibble, glue (>= 1.3.0)

RoxygenNote 6.1.1

Suggests testthat, knitr, rmarkdown, pacman, covr, spelling

SystemRequirements C++11

VignetteBuilder knitr

URL https://rorynolan.github.io/strex/

BugReports https://github.com/rorynolan/strex/issues

Language en-US

NeedsCompilation yes

Author Rory Nolan [aut, cre] (<https://orcid.org/0000-0002-5239-4043>)

Maintainer Rory Nolan <rorynoolan@gmail.com>

Repository CRAN

Date/Publication 2018-11-30 15:30:03 UTC

R topics documented:

`currency` *Get the currencies of numbers within a string.*

Description

The currency of a number is defined as the character coming before the number in the string. If nothing comes before (i.e. if the number is the first thing in the string), the currency is the empty string, similarly the currency can be a space, comma or any manner of thing.

- `get_currency` takes a string and returns the currency of the first number therein. It is vectorized over string.
- `get_currencies` takes a string and returns the currencies of all of the numbers within that string. It is not vectorized.

Usage

```
str_get_currencies(string)

str_get_currency(string)
```

Arguments

string	A string for get_currencies() and a character vector for get_currency().

Details

These functions do not allow for leading decimal points.

Value

- get_currency returns a character vector.
- get_currencies returns a data frame with one column for the currency symbol and one for the amount.

Examples

```
str_get_currencies("35.00 $1.14 abc5 $3.8 77")
str_get_currency(c("ab3 13", "$1"))
```

---

strex *strex: extra string manipulation functions*

Description

There are some things that I wish were easier with the stringr or stringi packages. The foremost of these is the extraction of numbers from strings. stringr makes you figure out the regex for yourself; strex takes care of this for you. There are many more useful functionalities in strex. In particular, there's a match_arg() function which is more flexible than the base match.arg(). Contributions to this package are encouraged: it is intended as a miscellany of string manipulation functions which cannot be found in stringi or stringr.

References

Rory Nolan and Sergi Padilla-Parra (2017). filesstrings: An R package for file and string manipulation. The Journal of Open Source Software, 2(14). doi: 10.21105/joss.00260.

`str_after_nth` *Text before or after nth occurrence of pattern.*

Description

Extract the part of a string which is before or after the *n*th occurrence of a specified pattern, vectorized over the string. `n` can be negatively indexed. See 'Arguments'.

Usage

```
str_after_nth(strings, pattern, n)

str_after_first(strings, pattern)

str_after_last(strings, pattern)

str_before_nth(strings, pattern, n)

str_before_first(strings, pattern)

str_before_last(strings, pattern)
```

Arguments

`strings`	A character vector.
`pattern`	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at `stringr::str_locate()`). If this has length >1 its length must be the same as that of `string`.
`n`	A natural number to identify the nth occurrence (defaults to first (`n = 1`)). This can be negatively indexed, so if you wish to select the *last* occurrence, you need `n = -1`, for the second-last, you need `n = -2` and so on.

Details

- `str_after_first(...)` is just `str_after_nth(..., n = 1)`.
- `str_after_last(...)` is just `str_after_nth(..., n = -1)`.
- `str_before_first(...)` is just `str_before_nth(..., n = 1)`.
- `str_before_last(...)` is just `str_before_nth(..., n = -1)`.

Value

A character vector of the desired strings.

Examples

```
string <- "ab..cd..de..fg..h"
str_after_nth(string, "\\.\\.", 3)
str_before_nth(string, "e", 1)
str_before_nth(string, "\\.", -3)
str_before_nth(string, ".", -3)
str_before_nth(rep(string, 2), fixed("."), -3)
```

`str_alphord_nums` *Make string numbers comply with alphabetical order.*

Description

If strings are numbered, their numbers may not *comply* with alphabetical order, i.e. "abc2" comes after "abc10" in alphabetical order. We might (for whatever reason) wish to change them such that they come in the order *that we would like*. This function alters the strings such that they comply with alphabetical order, so here "abc2" would be renamed to "abc02". It works on file names with more than one number in them e.g. "abc01def3" (a string with 2 numbers). All the file names that it works on must have the same number of numbers, and the non-number bits must be the same.

Usage

```
str_alphord_nums(strings)
```

Arguments

`strings`	A vector of strings.

Examples

```
strings <- paste0("abc", 1:12)
strings
str_alphord_nums(strings)
str_alphord_nums(c("abc9def55", "abc10def7"))
str_alphord_nums(c("01abc9def55", "5abc10def777", "99abc4def4"))
str_alphord_nums(1:10)

## Not run:
str_alphord_nums(c("abc9def55", "abc10xyz7"))
## End(Not run)
```

str_before_last_dot *Get the part of a string before the last period.*

Description

This is usually used to get the part of a file name that doesn't include the file extension. It is vectorized over `string`. If there is no period in `string`, the input is returned.

Usage

```
str_before_last_dot(string)
```

Arguments

`string`	A character vector.

Value

A character vector.

Examples

```
str_before_last_dot(c("spreadsheet1.csv", "doc2.doc", ".R"))
```

str_can_be_numeric *Check if a string could be considered as numeric.*

Description

After padding is removed, could the input string be considered to be numeric, i.e. could it be coerced to numeric. This function is vectorized over its one argument.

Usage

```
str_can_be_numeric(string)
```

Arguments

`string`	A character vector.

Value

A character vector. `TRUE` if the argument can be considered to be numeric or `FALSE` otherwise.

Examples

```
str_can_be_numeric("3")
str_can_be_numeric("5 ")
str_can_be_numeric(c("1a", "abc"))
```

`str_elem` *Extract a single character from a string, using its index.*

Description

If the element does not exist, this function returns the empty string.

Usage

```
str_elem(string, index)
```

Arguments

`string`	A string.
`index`	An integer. Negative indexing is allowed as in `stringr::str_sub()`.

Value

A one-character string.

Examples

```
str_elem(c("abcd", "xyz"), 3)
str_elem("abcd", -2)
```

`str_extract_non_numerics`
Extract non-numbers from a string.

Description

`str_extract_non_numerics` extracts the bits of the string that aren't extracted by `extract_numbers`. `str_nth_non_numeric` is a convenient wrapper for `str_extract_non_numerics`, allowing you to choose which number you want. Please run the examples at the bottom of this page to ensure that you understand how these functions work, and their limitations. These functions are vectorized over `string`.

Usage

```
str_extract_non_numerics(string, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)

str_nth_non_numeric(string, n, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)

str_first_non_numeric(string, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)

str_last_non_numeric(string, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)
```

Arguments

`string`	A string.
`decimals`	Do you want to include the possibility of decimal numbers (`TRUE`) or not (`FALSE`, the default).
`leading_decimals`	Do you want to allow a leading decimal point to be the start of a number?
`negs`	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
`n`	The index of the number (or non-numeric) that you seek. Negative indexing is allowed i.e. `n = 1` (the default) will give you the first number (or non-numeric) whereas `n = -1` will give you the last number (or non-numeric), `n = -2` will give you the second last number and so on. The function is vectorized over this argument.

Details

- `str_first_non_numeric(...)` is just `str_nth_non_numeric(..., n = 1)`.
- `str_last_non_numeric(...)` is just `str_nth_non_numeric(..., n = -1)`.

Examples

```
str_extract_non_numerics("abc123abc456")
str_extract_non_numerics("abc1.23abc456")
str_extract_non_numerics("abc1.23abc456", decimals = TRUE)
str_extract_non_numerics("abc1..23abc456", decimals = TRUE)
str_extract_non_numerics("abc1..23abc456", decimals = TRUE,
                    leading_decimals = TRUE)
str_extract_non_numerics(c("-123abc456", "ab1c"))
str_extract_non_numerics("-123abc456", negs = TRUE)
str_extract_non_numerics("--123abc456", negs = TRUE)
str_extract_non_numerics("--123abc456", negs = TRUE)
str_nth_non_numeric("--123abc456", 1)
str_nth_non_numeric("--123abc456", -2)
```

`str_extract_numbers` *Extract numbers from a string.*

Description

`str_extract_numbers` extracts the numbers (or non-numbers) from a string where decimals are optionally allowed. `str_nth_number` is a convenient wrapper for `str_extract_numbers`, allowing you to choose which number you want. Please run the examples at the bottom of this page to ensure that you understand how these functions work, and their limitations. These functions are vectorized over `string`.

Usage

```
str_extract_numbers(string, leave_as_string = FALSE, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)

str_nth_number(string, n, leave_as_string = FALSE, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)

str_first_number(string, leave_as_string = FALSE, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)

str_last_number(string, leave_as_string = FALSE, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE)
```

Arguments

`string`	A string.
`leave_as_string`	Do you want to return the number as a string (`TRUE`) or as numeric (`FALSE`, the default)?
`decimals`	Do you want to include the possibility of decimal numbers (`TRUE`) or not (`FALSE`, the default).
`leading_decimals`	Do you want to allow a leading decimal point to be the start of a number?
`negs`	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
`n`	The index of the number (or non-numeric) that you seek. Negative indexing is allowed i.e. `n = 1` (the default) will give you the first number (or non-numeric) whereas `n = -1` will give you the last number (or non-numeric), `n = -2` will give you the second last number and so on. The function is vectorized over this argument.

Details

If any part of a string contains an ambiguous number (e.g. `1.2.3` would be ambiguous if `decimals = TRUE` (but not otherwise)), the value returned for that string will be `NA`. Note that these functions do not know about scientific notation (e.g. `1e6` for 1000000).

- `str_first_number(...)` is just `str_nth_number(..., n = 1)`.
- `str_last_number(...)` is just `str_nth_number(..., n = -1)`.

Value

For `str_extract_numbers` and `extract_non_numerics`, a list of numeric or character vectors, one list element for each element of `string`. For `str_nth_number` and `nth_non_numeric`, a vector the same length as `string` (as in `length(string)`, not `nchar(string)`).

Examples

```
str_extract_numbers(c("abc123abc456", "abc1.23abc456"))
str_extract_numbers(c("abc1.23abc456", "abc1..23abc456"), decimals = TRUE)
str_extract_numbers("abc1..23abc456", decimals = TRUE)
str_extract_numbers("abc1..23abc456", decimals = TRUE, leading_decimals = TRUE)
str_extract_numbers("abc1..23abc456", decimals = TRUE, leading_decimals = TRUE,
               leave_as_string = TRUE)
str_extract_numbers("-123abc456")
str_extract_numbers("-123abc456", negs = TRUE)
str_extract_numbers("--123abc456", negs = TRUE)
str_extract_numbers(c(rep("abc1.2.3", 2), "a1b2.2.3", "e5r6"), decimals = TRUE)
str_extract_numbers("ab.1.2", decimals = TRUE, leading_decimals = TRUE)
str_nth_number("abc1.23abc456", 2:3)
str_nth_number("abc1.23abc456", 2, decimals = TRUE)
str_nth_number("-123abc456", -2, negs = TRUE)
```

---

str_give_ext *Ensure a file name has the intended extension.*

Description

Say you want to ensure a name is fit to be the name of a csv file. Then, if the input doesn't end with ".csv", this function will tack ".csv" onto the end of it. This is vectorized over the first argument.

Usage

```
str_give_ext(string, ext, replace = FALSE)
```

Arguments

`string`	The intended file name.
`ext`	The intended file extension (with or without the ".").
`replace`	If the file has an extension already, replace it (or append the new extension name)?

Value

A string: the file name in your intended form.

Examples

```
str_give_ext(c("abc", "abc.csv"), "csv")
str_give_ext("abc.csv", "pdf")
str_give_ext("abc.csv", "pdf", replace = TRUE)
```

---

str_locate_braces *Locate the braces in a string.*

Description

Give the positions of (,), [,], {, } within a string.

Usage

```
str_locate_braces(string)
```

Arguments

string	A character vector

Value

A list of data frames, one for each member of the string character vector. Each data frame has a "position" and "brace" column which give the positions and types of braces in the given string.

Examples

```
str_locate_braces(c("a{](kkj)})", "ab(]c{}"))
```

---

str_locate_nth *Locate the indices of the nth instance of a pattern.*

Description

The nth instance of an pattern will cover a series of character indices. These functions tell you which indices those are.

Usage

```
str_locate_nth(string, pattern, n)

str_locate_first(string, pattern)

str_locate_last(string, pattern)
```

Arguments

string	A character vector. These functions are vectorized over this argument.
pattern	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at stringr::str_locate()). If this has length >1 its length must be the same as that of string.
n	Then n for the nth instance of the pattern.

Details

- str_locate_first(...) is just str_locate_nth(..., n = 1).
- str_locate_last(...) is just str_locate_nth(..., n = -1).

Value

A two-column matrix. The ith row of this matrix gives the start and end indices of the nth instance of pattern in the ith element of string.

Examples

```
str_locate_nth(c("abcdabcxyz", "abcabc"), "abc", 2)
```

---

str_match_arg *Argument Matching.*

Description

Match arg against a series of candidate choices where NULL means take the first one. arg *matches* an element of choices if arg is a prefix of that element.

Usage

```
str_match_arg(arg, choices, index = FALSE, several_ok = FALSE,
  ignore_case = FALSE)

match_arg(arg, choices, index = FALSE, several_ok = FALSE,
  ignore_case = FALSE)
```

Arguments

arg	A character vector (of length one unless several_ok = TRUE).
choices	A character vector of candidate values.
index	Return the index of the match rather than the match itself? Default no.
several_ok	Allow arg to have length greater than one to match several arguments at once? Default no.
ignore_case	Ignore case while matching. Default no. If this is TRUE, the returned value is the matched element of choices (with its original casing).

Details

ERRORs are thrown when a match is not made and where the match is ambiguous. However, sometimes ambiguities are inevitable. Consider the case where `choices = c("ab", "abc")`, then there's no way to choose `"ab"` because `"ab"` is a prefix for `"ab"` and `"abc"`. If this is the case, you need to provide a full match, i.e. using `arg = "ab"` will get you `"ab"` without an error, however `arg = "a"` will throw an ambiguity error.

This function inspired by `RSAGA::match.arg.ext()`. Its behaviour is almost identical (the difference is that `RSAGA::match.arg.ext(..., ignore.case = TRUE)` guarantees that the function returns strings in all lower case, but that is not so with `filesstrings::match_arg(..., ignore_case = TRUE)`) but RSAGA is a heavy package to depend upon so `filesstrings::match_arg()` might be handy for package developers.

This function is designed to be used inside of other functions. It's fine to use it for other purposes, but the error messages might be a bit weird.

Examples

```
choices <- c("Apples", "Pears", "Bananas", "Oranges")
match_arg(NULL, choices)
match_arg("A", choices)
match_arg("B", choices, index = TRUE)
match_arg(c("a", "b"), choices, several_ok = TRUE, ignore_case = TRUE)
match_arg(c("b", "a"), choices, ignore_case = TRUE, index = TRUE,
          several_ok = TRUE)
```

---

str_nth_number_after_mth

Find the nth number after the mth occurrence of a pattern.

Description

Given a string, a pattern and natural numbers `n` and `m`, find the `n`th number after the `m`th occurrence of the pattern.

Usage

```
str_nth_number_after_mth(string, pattern, n, m, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_nth_number_after_first(string, pattern, n, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_nth_number_after_last(string, pattern, n, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_first_number_after_mth(string, pattern, m, decimals = FALSE,
```

```
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_last_number_after_mth(string, pattern, m, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_first_number_after_first(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_first_number_after_last(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_last_number_after_first(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_last_number_after_last(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)
```

Arguments

`string`	A character vector.
`pattern`	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at `stringr::str_locate()`). If this has length >1 its length must be the same as that of `string`.
`n, m`	Natural numbers.
`decimals`	Do you want to include the possibility of decimal numbers (`TRUE`) or not (`FALSE`, the default).
`leading_decimals`	Do you want to allow a leading decimal point to be the start of a number?
`negs`	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
`leave_as_string`	Do you want to return the number as a string (`TRUE`) or as numeric (`FALSE`, the default)?

Value

A numeric vector.

Examples

```
string <- c("abc1abc2abc3abc4abc5abc6abc7abc8abc9",
            "abc1def2ghi3abc4def5ghi6abc7def8ghi9")
str_nth_number_after_mth(string, "abc", 1, 3)
str_nth_number_after_mth(string, "abc", 2, 3)
str_nth_number_after_first(string, "abc", 2)
str_nth_number_after_last(string, "abc", -1)
str_first_number_after_mth(string, "abc", 2)
str_last_number_after_mth(string, "abc", 1)
```

```
str_first_number_after_first(string, "abc")
str_first_number_after_last(string, "abc")
str_last_number_after_first(string, "abc")
str_last_number_after_last(string, "abc")
```

`str_nth_number_before_mth` *Find the nth number before the mth occurrence of a pattern.*

Description

Given a string, a pattern and natural numbers `n` and `m`, find the `n`th number that comes before the `m`th occurrence of the pattern.

Usage

```
str_nth_number_before_mth(string, pattern, n, m, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_nth_number_before_first(string, pattern, n, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_nth_number_before_last(string, pattern, n, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_first_number_before_mth(string, pattern, m, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_last_number_before_mth(string, pattern, m, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_first_number_before_first(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_first_number_before_last(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_last_number_before_first(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)

str_last_number_before_last(string, pattern, decimals = FALSE,
  leading_decimals = FALSE, negs = FALSE, leave_as_string = FALSE)
```

Arguments

`string`	A character vector.

pattern	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at stringr::str_locate()). If this has length >1 its length must be the same as that of string.
n, m	Natural numbers.
decimals	Do you want to include the possibility of decimal numbers (TRUE) or not (FALSE, the default).
leading_decimals	Do you want to allow a leading decimal point to be the start of a number?
negs	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).
leave_as_string	Do you want to return the number as a string (TRUE) or as numeric (FALSE, the default)?

Value

A numeric vector.

Examples

```
string <- c("abc1abc2abc3abc4def5abc6abc7abc8abc9",
            "abc1def2ghi3abc4def5ghi6abc7def8ghi9")
str_nth_number_before_mth(string, "def", 1, 1)
str_nth_number_before_mth(string, "abc", 2, 3)
str_nth_number_before_first(string, "def", 2)
str_nth_number_before_last(string, "def", -1)
str_first_number_before_mth(string, "abc", 2)
str_last_number_before_mth(string, "def", 1)
str_first_number_before_first(string, "def")
str_first_number_before_last(string, "def")
str_last_number_before_first(string, "def")
str_last_number_before_last(string, "def")
```

---

str_paste_elems *Extract bits of a string and paste them together*

Description

Extract characters - specified by their indices - from a string and paste them together

Usage

```
str_paste_elems(string, indices)
```

Arguments

string	A string.
indices	A numeric vector of positive integers detailing the indices of the characters of string that we wish to paste together.

Value

A string.

Examples

```
str_paste_elems("abcdef", c(2, 5:6))
```

---

str_remove_quoted *Remove the quoted parts of a string.*

Description

If any parts of a string are quoted (between quotation marks), remove those parts of the string, including the quotes. Run the examples and you'll know exactly how this function works.

Usage

```
str_remove_quoted(string)
```

Arguments

string	A character vector.

Value

A character vector.

Examples

```
string <- "\"abc\"67a\'dk\'f"
cat(string)
str_remove_quoted(string)
```

---

str_singleize *Remove back-to-back duplicates of a pattern in a string.*

Description

If a string contains a given pattern duplicated back-to-back a number of times, remove that duplication, leaving the pattern appearing once in that position (works if the pattern is duplicated in different parts of a string, removing all instances of duplication). This is vectorized over string and pattern.

Usage

```
str_singleize(string, pattern)
```

Arguments

string	A character vector. The string(s) to be purged of duplicates.
pattern	A character vector. Pattern(s) specified like the pattern(s) in the stringr package (e.g. look at stringr::str_locate()). If this has length >1 its length must be the same as that of string.

Value

The string with the duplicates fixed.

Examples

```
str_singleize("abc//def", "/")
str_singleize("abababcabab", "ab")
str_singleize(c("abab", "cdcd"), "cd")
str_singleize(c("abab", "cdcd"), c("ab", "cd"))
```

---

str_split_by_nums *Split a string by its numeric characters.*

Description

Break a string wherever you go from a numeric character to a non-numeric or vice-versa.

Usage

```
str_split_by_nums(string, decimals = FALSE, leading_decimals = FALSE,
  negs = FALSE)
```

Arguments

string	A string.
decimals	Do you want to include the possibility of decimal numbers (TRUE) or not (FALSE, the default).
leading_decimals	Do you want to allow a leading decimal point to be the start of a number?
negs	Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).

Examples

```
str_split_by_nums(c("abc123def456.789gh", "a1b2c344"))
str_split_by_nums("abc123def456.789gh", decimals = TRUE)
str_split_by_nums("22")
```

str_split_camel_case *Split a string based on CamelCase*

Description

Vectorized over `string`.

Usage

```
str_split_camel_case(string, lower = FALSE)
```

Arguments

`string`	A character vector.
`lower`	Do you want the output to be all lower case (or as is)?

Value

A list of character vectors, one list element for each element of `string`.

References

Adapted from Ramnath Vaidyanathan's answer at http://stackoverflow.com/questions/8406974/splitting-camelcase-in-r.

Examples

```
str_split_camel_case(c("RoryNolan", "NaomiFlagg", "DepartmentOfSillyHats"))
```

str_to_vec *Convert a string to a vector of characters*

Description

Go from a string to a vector whose ith element is the ith character in the string.

Usage

```
str_to_vec(string)
```

Arguments

`string`	A string.

Value

A character vector.

Examples

```
str_to_vec("abcdef")
```

`str_trim_anything` *Trim something other than whitespace*

Description

The `stringi` and `stringr` packages let you trim whitespace, but what if you want to trim something else from either (or both) side(s) of a string? This function lets you select which pattern to trim and from which side(s).

Usage

```
str_trim_anything(string, pattern, side = "both")
```

Arguments

`string`	A string.
`pattern`	A string. The pattern to be trimmed (*not* interpreted as regular expression). So to trim a period, use `char = "."` and not `char = "\\."`).
`side`	Which side do you want to trim from? `"both"` is the default, but you can also have just either `"left"` or `"right"` (or optionally the shortened `"b"`, `"l"` and `"r"`).

Value

A string.

Examples

```
str_trim_anything("..abcd.", ".", "left")
str_trim_anything("-ghi--", "-")
str_trim_anything("-ghi--", "--")
```

Index