Title Extra String Manipulation Functions
Version 1.2.0
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'.
License GPL-3
URL https://rorynolan.github.io/strex,
https://github.com/rorynolan/strex
BugReports https://github.com/rorynolan/strex/issues
Depends R (>= 3.1), stringr (>= 1.2.0)
Imports checkmate (>= 1.9.3), glue (>= 1.3.0), magrittr (>= 1.5),
    matrixStats (>= 0.50.0), ore (>= 1.4.0), processx (>= 3.3.1),
    Rcpp (>= 1.0.1), rlang (>= 0.3.3), stringi (>= 1.3.1),
    tibble (>= 2.0.1)
Suggests covr, knitr, pacman, purrr, rmarkdown, spelling, testthat (>=
    2.1.0)
LinkingTo Rcpp (>= 1.0.1)
VignetteBuilder knitr
Biarch TRUE
Encoding UTF-8
Language en-US
LazyData true
RoxygenNote 6.1.1
SystemRequirements C++11
NeedsCompilation yes
before-and-after

Extract text before or after nth occurrence of pattern.

Description

Extract the part of a string which is before or after the nth occurrence of a specified pattern, vectorized over the string.
Usage

str_after_nth(string, pattern, n)
str_after_first(string, pattern)
str_after_last(string, pattern)
str_before_nth(string, pattern, n)
str_before_first(string, pattern)
str_before_last(string, pattern)

Arguments

string A character vector.
pattern The pattern to look for.
The default interpretation is a regular expression, as described in stringi::stringi-search-regex.
To match a without regular expression (i.e. as a human would), use coll(). For details see stringr::regex().
n A vector of integerish values. Must be either length 1 or have length equal to
the length of string. Negative indices count from the back: while n = 1 and
n = 2 correspond to first and second, n = -1 and n = -2 correspond to last and
second-last. n = 0 will return NA.

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.

See Also

Other bisectors: str_before_last_dot

Examples

string <- "abxxcdxxdxxfgxxh"
str_after_nth(string, "xx", 3)
str_before_nth(string, "e", 1:2)
str_before_nth(string, "xx", -3)
str_before_nth(string, ".", -3)
str_before_nth(rep(string, 2), ".x", -3)
str_before_first(string, "d")
str_before_last(string, "x")
string <- c("abc", "xyz.zyx")
str_after_first(string, ".") # using regex
str_after_first(string, coll(".")) # using human matching
str_after_last(c("xy", "xz"), "x")

currency

<table>
<thead>
<tr>
<th>currency</th>
<th>Extract currency amounts from a string.</th>
</tr>
</thead>
</table>

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.

Usage

str_extract_currencies(string)

str_nth_currency(string, n)

str_first_currency(string)

str_last_currency(string)

Arguments

string A character vector.

n A vector of integerish values. Must be either length 1 or have length equal to the length of string. Negative indices count from the back: while n = 1 and n = 2 correspond to first and second, n = -1 and n = -2 correspond to last and second-last. n = 0 will return NA.

Details

These functions are vectorized over string and n.

str_extract_currencies() extracts all currency amounts.

str_nth_currency() just gets the nth currency amount from each string. str_first_currency(string) and str_last_currency(string) are just wrappers for str_nth_currency(string, n = 1) and str_nth_currency(string, n = -1).

"-$2.00" and "$-2.00" are interpreted as negative two dollars.

If you request e.g. the 5th currency amount but there are only 3 currency amounts, you get an amount and currency symbol of NA.
Value

A tibble with 4 columns: `string_num`, `string`, `curr_sym` and `amount`. Every extracted currency amount gets its own row in the tibble detailing the string number and string that it was extracted from, the currency symbol and the amount.

Examples

```r
string <- c("ab3 13", "$1", "35.00 $1.14", "abc5 $3.8", "stuff")
str_extract_currencies(string)
str_nth_currency(string, n = 2)
str_nth_currency(string, n = -2)
str_nth_currency(string, c(1, -2, 1, 2, -1))
str_first_currency(string)
str_last_currency(string)
```

---

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


---

str_alphord_nums

Make string numbers comply with alphabetical order.

Description

If strings are numbered, their numbers may not comply with alphabetical order, e.g. "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 strings in the character vector `string` must have the same number of numbers, and the non-number bits must be the same.
### str_alphord_nums

#### Usage

\`str_alphord_nums(string)`

#### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A character vector.</td>
</tr>
</tbody>
</table>

#### Value

A character vector.

#### Examples

```r
string <- paste0("abc", 1:12)
print(string)
str_alphord_nums(string)
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")) # error
## End(Not run)
```

### str_before_last_dot

Extract 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A character vector.</td>
</tr>
</tbody>
</table>

#### Value

A character vector.

#### See Also

Other bisectors: `before-and-after`
str_can_be_numeric

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 logical vector.

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. This is consistent with stringr::str_sub().
This function is vectorised over both arguments.

Usage

str_elem(string, index)

Arguments

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

A one-character string.

See Also

Other single element extractors: \texttt{str_elems, str_paste elems}

Examples

\begin{verbatim}
str_elem(c("abcd", "xyz"), 3)
str_elem("abcd", -2)
\end{verbatim}

\begin{verbatim}
str elems (string, indices, byrow = TRUE)
\end{verbatim}

\textbf{Description}

Efficiently extract several elements from a string. See \texttt{str elem()} for extracting single elements. This function is vectorized over the first argument.

\textbf{Usage}

\begin{verbatim}
str elems(string, indices, byrow = TRUE)
\end{verbatim}

\textbf{Arguments}

\begin{description}
\item[string] A character vector.
\item[indices] A vector of integerish values. Negative indexing is allowed as in \texttt{stringr::str_sub()}.\item[byrow] Should the elements be organised in the matrix with one row per string (byrow = \texttt{TRUE}, the default) or one column per string (byrow = \texttt{FALSE}). See examples if you don’t understand.
\end{description}

\textbf{Value}

A character matrix.

\textbf{See Also}

Other single element extractors: \texttt{str elem, str paste elems}

\textbf{Examples}

\begin{verbatim}
string <- c("abc", "def", "ghi", "vwxz")
str elems(string, 1:2)
str elems(string, 1:2, byrow = FALSE)
str elems(string, c(1, 2, 3, 4, -1))
\end{verbatim}
\textit{str\_extract\_non\_numerics}

\textit{Extract non-numbers from a string.}

**Description**

Extract the non-numeric bits of a string where numbers are optionally defined with decimals, scientific notation and commas (as separators, not as an alternative to the decimal point).

**Usage**

\[
\text{str\_extract\_non\_numerics}(\text{string, decimals = FALSE, leading\_decimals = decimals, negs = FALSE, sci = FALSE, commas = FALSE})
\]

**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).
- **sci**: Make the search aware of scientific notation e.g. 2e3 is the same as 2000.
- **commas**: Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).

**Details**

- \text{str\_first\_non\_numeric}(\ldots) is just \text{str\_nth\_non\_numeric}(\ldots, n = 1).
- \text{str\_last\_non\_numeric}(\ldots) is just \text{str\_nth\_non\_numeric}(\ldots, n = -1).

**See Also**

Other non-numeric extractors: \text{str\_nth\_non\_numeric}

**Examples**

\[
\text{strings} \leftarrow c(
\quad \text{"abc123def456", "abc-0.12def.345", "abc.12e4def34.5e9",}
\quad \text{"abc1,100def1,230.5", "abc1,100e3,215def4e1,000"}
\)
\text{str\_extract\_non\_numerics(strings)}
\text{str\_extract\_non\_numerics(strings, decimals = TRUE, leading\_decimals = FALSE)}
\text{str\_extract\_non\_numerics(strings, decimals = TRUE)}
str_extract_non_numerics(strings, commas = TRUE)
str_extract_non_numerics(strings,
    decimals = TRUE, leading_decimals = TRUE,
    sci = TRUE)
str_extract_non_numerics(strings,
    decimals = TRUE, leading_decimals = TRUE,
    sci = TRUE, commas = TRUE, negs = TRUE)
str_extract_non_numerics(c("22", "1.2.3"), decimals = TRUE)

str_extract_numbers Extract numbers from a string.

Description

Extract the numbers from a string, where decimals, scientific notation and commas (as separators, not as an alternative to the decimal point) are optionally allowed.

Usage

str_extract_numbers(string, decimals = FALSE,
    leading_decimals = decimals, negs = FALSE, sci = FALSE,
    commas = FALSE, leave_as_string = 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).
sci Make the search aware of scientific notation e.g. 2e3 is the same as 2000.
commas Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).
leave_as_string Do you want to return the number as a string (TRUE) or as numeric (FALSE, the default)?
**str_extract_numbers**

**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 and a warning will be issued.

With scientific notation, it is assumed that the exponent is not a decimal number e.g. 2e2.4 is unacceptable. Commas, however, are acceptable in the exponent, so 2e1.100 is fine and equal to 2e1100 if the option to allow commas in numbers has been turned on.

Numbers outside the double precision floating point range (i.e. with absolute value greater than 1.797693e+308) are read as Inf (or -Inf if they begin with a minus sign). This is what base::as.numeric() does.

**Value**

For str_extract_numbers and str_extract_non_numerics, a list of numeric or character vectors, one list element for each element of string. For str_nth_number and str_nth_non_numeric, a numeric or character vector the same length as the vector string.

**See Also**

Other numeric extractors: str_nth_number_after_mth, str_nth_number_before_mth, str_nth_number

**Examples**

```r
strings <- c(
  "abc123def456", "abc-0.12def.345", "abc.12e4def34.5e9",
  "abc1,100def1,230.5", "abc1,100e3,215def4e1,000"
)
str_extract_numbers(strings)
str_extract_numbers(strings, decimals = TRUE)
str_extract_numbers(strings, decimals = TRUE, leading_decimals = TRUE)
str_extract_numbers(strings, commas = TRUE)
str_extract_numbers(strings,
  decimals = TRUE, leading_decimals = TRUE,
  sci = TRUE
)
str_extract_numbers(strings,
  decimals = TRUE, leading_decimals = TRUE,
  sci = TRUE, commas = TRUE, negs = TRUE
)
str_extract_numbers(strings,
  decimals = TRUE, leading_decimals = FALSE,
  sci = FALSE, commas = TRUE, leave_as_string = TRUE
)
str_extract_numbers(c("22", "1.2.3"), decimals = 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**

```r
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**

```r
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
**str_locate_nth**

**Value**

A **tibble** with 4 columns: `string_num`, `string`, `position` and `brace`. Every extracted brace amount gets its own row in the tibble detailing the string number and string that it was extracted from, the position in its string and the brace.

**See Also**

Other locators: **str_locate_nth**

**Examples**

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

---

**str_locate_nth**

Locate the indices of the **n**th instance of a pattern.

**Description**

The **n**th instance of an pattern will cover a series of character indices. These functions tell you which indices those are. These functions are vectorised over all arguments.

**Usage**

```r
str_locate_nth(string, pattern, n)
str_locate_first(string, pattern)
str_locate_last(string, pattern)
```

**Arguments**

- `string` A character vector.
- `pattern` The pattern to look for.
  - The default interpretation is a regular expression, as described in `stringi::stringi-search-regex`.
  - To match a without regular expression (i.e. as a human would), use `coll()`. For details see `stringr::regex()`.
- `n` A vector of integerish values. Must be either length 1 or have length equal to the length of `string`. Negative indices count from the back: while `n = 1` and `n = 2` correspond to first and second, `n = -1` and `n = -2` correspond to last and second-last. `n = 0` will return NA.

**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 $i$th row of this matrix gives the start and end indices of the $n$th instance of pattern in the $i$th element of string.

See Also
Other locators: str_locate_braces

Examples
str_locate_nth(c("abcdabcxyz", "abcabc"), "abc", 2)
str_locate_nth(
  c("This old thing.", "That beautiful thing there."),
  "\w+", c(2, -2)
)
str_locate_nth("abc", "b", c(0, 1, 1, 2))
str_locate_first("abcxyzabc", "abc")
str_locate_last("abcxyzabc", "abc")

---

str_match_arg  Argument Matching.

Description
Match arg against a series of candidate choices. arg matches an element of choices if arg is a prefix of that element.

Usage
str_match_arg(arg, choices = NULL, index = FALSE, several_ok = FALSE, ignore_case = FALSE)
match_arg(arg, choices = NULL, 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?
- **several_ok**  Allow arg to have length greater than one to match several arguments at once?
- **ignore_case**  Ignore case while matching. 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.

When `choices` is `NULL`, the choices are obtained from a default setting for the formal argument `arg` of the function from which `str_match_arg` was called. This is consistent with `base::match.arg()`. See the examples for details.

When `arg` and `choices` are identical and `several_ok = FALSE`, the first element of `choices` is returned. This is consistent with `base::match.arg()`.

This function inspired by `RSAGA::match.arg.ext()`. Its behaviour is almost identical (the difference is that `RSAGA::match.arg.ext(...,ignore.case = TRUE)` always returns in all lower case; `strex::match_arg(...,ignore_case = TRUE)` ignores case while matching but returns the element of `choices` in its original case). `RSAGA` is a heavy package to depend upon so `strex::match_arg()` is 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**

```r
c <- c("Apples", "Pears", "Bananas", "Oranges")
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)
myword <- function(w = c("abacus", "baseball", "candy")) {
  w <- match_arg(w)
  w
}
myword("b")
myword()
myword <- function(w = c("abacus", "baseball", "candy")) {
  w <- match_arg(w, several_ok = TRUE)
  w
}
myword("c")
myword()
```

---

**str_nth_non_numeric**

Extract the nth non-numeric substring from a string.
Description

Extract the nth non-numeric bit of a string where numbers are optionally defined with decimals, scientific notation and commas (as separators, not as an alternative to the decimal point).

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

Usage

```r
str_nth_non_numeric(string, n, decimals = FALSE,
leading_decimals = decimals, negs = FALSE, sci = FALSE,
commas = FALSE)

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

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

Arguments

- `string` A string.
- `n` A vector of integerish values. Must be either length 1 or have length equal to the length of `string`. Negative indices count from the back: while `n = 1` and `n = 2` correspond to first and second, `n = -1` and `n = -2` correspond to last and second-last. `n = 0` will return `NA`.
- `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).
- `sci` Make the search aware of scientific notation e.g. `2e3` is the same as `2000`.
- `commas` Allow comma separators in numbers (i.e. interpret `1,100` as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).

See Also

Other non-numeric extractors: `str_extract_non_numerics`

Examples

```r
strings <- c(
  "abc123def456", "abc-0.12def.345", "abc.12e4def34.5e9",
  "abc1,100def1,230.5", "abc1,100e3,215def4e1,000"
)
**str_nth_number**

Extract the nth number from a string.

**Description**

Extract the nth number from a string, where decimals, scientific notation and commas (as separators, not as an alternative to the decimal point) are optionally allowed.

**Usage**

```r
str_nth_number(string, n, decimals = FALSE,
               leading_decimals = decimals, negs = FALSE, sci = FALSE,
               commas = FALSE, leave_as_string = FALSE)
```

```r
str_first_number(string, decimals = FALSE, leading_decimals = decimals,
                 negs = FALSE, sci = FALSE, commas = FALSE,
                 leave_as_string = FALSE)
```

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

**Arguments**

- **string**
  A string.
- **n**
  A vector of integerish values. Must be either length 1 or have length equal to the length of string. Negative indices count from the back: while n = 1 and n = 2 correspond to first and second, n = -1 and n = -2 correspond to last and second-last. n = 0 will return NA.
- **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).

sci  Make the search aware of scientific notation e.g. 2e3 is the same as 2000.

commas  Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).

leave_as_string  Do you want to return the number as a string (TRUE) or as numeric (FALSE, the default)?

Details

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

For a detailed explanation of the number extraction, see str_extract_numbers().

Value

A numeric vector (or a character vector if leave_as_string = TRUE).

See Also

Other numeric extractors: str_extract_numbers, str_nth_number_after_mth, str_nth_number_before_mth

Examples

strings <- c(
  "abc123def456", "abc-0.12def.345", "abc.12e4def34.5e9",
  "abc1,100def1,230.5", "abc1,100e3,215def4e1,000"
)
str_nth_number(strings, n = 2)
str_nth_number(strings, n = -2, decimals = TRUE)
str_first_number(strings, decimals = TRUE, leading_decimals = TRUE)
str_last_number(strings, commas = TRUE)
str_nth_number(strings,
  n = 1, decimals = TRUE, leading_decimals = TRUE,
  sci = TRUE)
str_first_number(strings,
  decimals = TRUE, leading_decimals = TRUE,
  sci = TRUE, commas = TRUE, negs = TRUE)
str_last_number(strings,
  decimals = TRUE, leading_decimals = FALSE,
  sci = FALSE, commas = TRUE, negs = TRUE, leave_as_string = TRUE)
str_first_number(c("22", "1.2.3"), decimals = TRUE)
*str_nth_number_after_mth*

Find the \( n \)th number after the \( m \)th 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**

\[
\text{str_nth_number_after_mth}(\text{string}, \text{pattern}, n, m, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_nth_number_after_first}(\text{string}, \text{pattern}, n, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_nth_number_after_last}(\text{string}, \text{pattern}, n, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_first_number_after_mth}(\text{string}, \text{pattern}, m, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_last_number_after_mth}(\text{string}, \text{pattern}, m, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_first_number_after_first}(\text{string}, \text{pattern}, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_first_number_after_last}(\text{string}, \text{pattern}, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_last_number_after_first}(\text{string}, \text{pattern}, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]

\[
\text{str_last_number_after_last}(\text{string}, \text{pattern}, \text{decimals} = \text{FALSE}, \text{leading_decimals} = \text{decimals}, \text{nega} = \text{FALSE}, \text{sci} = \text{FALSE}, \text{commas} = \text{FALSE}, \text{leave_as_string} = \text{FALSE})
\]
Arguments

string A character vector.

pattern The pattern to look for.

The default interpretation is a regular expression, as described in stringi::stringi-search-regex.

To match a without regular expression (i.e. as a human would), use coll(). For details see stringr::regex().

n, m Vectors of integerish values. Must be either length 1 or have length equal to the length of string. Negative indices count from the back: while 1 and 2 correspond to first and second, -1 and -2 correspond to last and second-last. 0 will return NA.

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).

sci Make the search aware of scientific notation e.g. 2e3 is the same as 2000.

comas Allow comma separators in numbers (i.e. interpret 1,100 as a single number (one thousand one hundred) rather than two numbers (one and one hundred)).

leave_as_string Do you want to return the number as a string (TRUE) or as numeric (FALSE, the default)?

Value

A numeric or character vector.

See Also

Other numeric extractors: str_extract_numbers, str_nth_number_before_mth, str_nth_number

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 nth number that comes before the mth occurrence of the pattern.

Usage

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

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

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

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

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

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

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

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

str_last_number_before_last(string, pattern, decimals = FALSE, leading_decimals = decimals, negs = FALSE, sci = FALSE, commas = FALSE, leave_as_string = FALSE)
str_nth_number_before_mth

Arguments

string A character vector.
pattern The pattern to look for.

The default interpretation is a regular expression, as described in
stringi::stringi-search-regex.

To match a without regular expression (i.e. as a human would), use coll(). For
details see stringr::regex().
n Vectors of integerish values. Must be either length 1 or have length equal to
the length of string. Negative indices count from the back: while 1 and 2
correspond to first and second, -1 and -2 correspond to last and second-last. 0
will return NA.
m Vectors of integerish values. Must be either length 1 or have length equal to
the length of string. Negative indices count from the back: while 1 and 2
correspond to first and second, -1 and -2 correspond to last and second-last. 0
will return NA.
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).
sci Make the search aware of scientific notation e.g. 2e3 is the same as 2000.
commas Allow comma separators in numbers (i.e. interpret 1,100 as a single number
(one thousand one hundred) rather than two numbers (one and one hundred)).
leave_as_string Do you want to return the number as a string (TRUE) or as numeric (FALSE, the default)?

Value

A numeric or character vector.

See Also

Other numeric extractors: str_extract_numbers, str_nth_number_after_mth, str_nth_number

Examples

```r
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_paste_elems

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 single elements of a string and paste them together.

Description

This is a quick way around doing a call to str_elems() followed by a call of apply(..., paste).

Usage

str_paste_elems(string, indices, sep = "")

Arguments

string A character vector.
indices A vector of integerish values. Negative indexing is allowed as in stringr::str_sub().
sep A string. The separator for pasting string elements together.

Details

Elements that don’t exist e.g. element 5 of "abc" are ignored.

Value

A character vector.

See Also

Other single element extractors: str_elems, str_elem

Examples

string <- c("abc", "def", "ghi", "vwxyz")
str_paste_elems(string, 1:2)
str_paste_elems(string, c(1, 2, 3, 4, -1))
str_paste_elems("abc", c(1, 5, 55, 43, 3))
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.

See Also

Other removers: str_singleize, str_trim_anything

Examples

```r
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)
str_split_by_numbers

Arguments

string A character vector.

pattern The pattern to look for.

The default interpretation is a regular expression, as described in stringi::stringi-search-regex.
To match a without regular expression (i.e. as a human would), use col(). For details see stringr::regex().

Value
A character vector.

See Also
Other removers: str_remove_quoted, str_trim_anything

Examples

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

str_split_by_numbers 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. Keep the whole string, just split it up. Vectorised over string.

Usage

str_split_by_numbers(string, decimals = FALSE,
leading_decimals = FALSE, negs = FALSE, sci = FALSE,
commas = 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).
str_split_camel_case

sci
Make the search aware of scientific notation e.g. 2e3 is the same as 2000.

commas
Allow comma separators in numbers (i.e. interpret 1,100 as a single number
(one thousand one hundred) rather than two numbers (one and one hundred)).

Value
A list of character vectors.

See Also
Other splitters: str_split_camel_case

Examples
str_split_by_numbers(c("abc123def456.789gh", "a1b2c344"))
str_split_by_numbers("abc123def456.789gh", decimals = TRUE)
str_split_by_numbers(c("22", "1.2.3"), decimals = TRUE)

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.

See Also
Other splitters: str_split_by_numbers
**str_to_vec**

*Convert a string to a vector of characters*

**Description**

Go from a string to a vector whose $i$th element is the $i$th character in the string.

**Usage**

```r
str_to_vec(string)
```

**Arguments**

- `string` A character vector.

**Value**

A character vector.

**Examples**

```r
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**

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

string  A character vector.
pattern  The pattern to look for.
          The default interpretation is a regular expression, as described in stringi::stringi-
          search-regex.
          To match a without regular expression (i.e. as a human would), use coll(). For
          details see stringr::regex().
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.

See Also

Other removers: str_remove_quoted, str_singleize

Examples

```r
str_trim_anything("..abcd.", ".", "left")
str_trim_anything("..abcd.", coll(“.”), "left")
str_trim_anything("-ghi--", "-", "both")
str_trim_anything("-ghi--", "-")
str_trim_anything("-ghi--", "-", "right")
str_trim_anything("-ghi--", "-", "right")
str_trim_anything("-ghi--", "i-+")
```
Index

before-and-after, 2

coll(), 3, 13, 20, 22, 25, 28
currency, 4

match_arg (str_match_arg), 14

str_after_first (before-and-after), 2
str_after_last (before-and-after), 2
str_after_nth (before-and-after), 2
str_alphord_nums, 5
str_before_first (before-and-after), 2
str_before_last (before-and-after), 2
str_before_last_dot, 3, 6
str_before_nth (before-and-after), 2
str_can_be_numeric, 7
str_elem, 7, 8, 23
str_elems, 8, 8, 23

str_extract_currencies (currency), 4
str_extract_currencies(), 4
str_extract_non_numerics, 9, 16
str_extract_numbers, 10, 18, 20, 22
str_extract_numbers(), 18
str_first_currency (currency), 4
str_first_non_numeric
(str_nth_non_numeric), 15
str_first_number (str_nth_number), 17
str_first_number_after_first
(str_nth_number_after_mth), 19
str_first_number_after_last
(str_nth_number_after_mth), 19
str_first_number_after_mth
(str_nth_number_after_mth), 19
str_first_number_after_first
(str_nth_number_before_mth), 21
str_first_number_after_last
(str_nth_number_before_mth), 21

str_first_number_before_mth
(str_nth_number_before_mth), 21
str_last_currency (currency), 4
str_last_non_numeric
(str_nth_non_numeric), 15
str_last_number (str_nth_number), 17
str_last_number_after_first
(str_nth_number_after_mth), 19
str_last_number_after_last
(str_nth_number_after_mth), 19
str_last_number_after_mth
(str_nth_number_after_mth), 19
str_last_number_before_first
(str_nth_number_before_mth), 21
str_last_number_before_last
(str_nth_number_before_mth), 21
str_last_number_before_mth
(str_nth_number_before_mth), 21
str_locate_braces, 12, 14
str_locate_first (str_locate_nth), 13
str_locate_last (str_locate_nth), 13
str_locate_nth, 13, 13
str_match_arg, 14
str_nth_currency (currency), 4
str_nth_non_numeric, 9, 15
str_nth_number, 11, 17, 20, 22
str_nth_number_after_first
(str_nth_number_after_mth), 19
str_nth_number_after_last
(str_nth_number_after_mth), 19
str_nth_number_after_mth
(str_nth_number_after_mth), 19
str_nth_number_before_first
(str_nth_number_before_mth), 21
str_nth_number_before_last
(str_nth_number_before_mth), 21
str_nth_number_before_mth
(str_nth_number_before_mth), 21

str_remove_quoted, 24, 25, 28

29
str_singleize, 24, 24, 28
str_split_by_numbers, 25, 26
str_split_camel_case, 26, 26
str_to_vec, 27
str_trim_anything, 24, 25, 27
strex, 5
strex-package (strex), 5
stringi::stringi-search-regex, 3, 13, 20, 22, 25, 28
stringr::regex(), 3, 13, 20, 22, 25, 28
stringr::str_sub(), 7, 8, 23

tibble, 5, 13