Package ‘strex’

October 30, 2023

Title Extra String Manipulation Functions
Version 1.6.1
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
BugReports https://github.com/rorynolan/strex/issues
Depends R (>= 3.5), stringr (>= 1.5)
Imports checkmate (>= 1.9.3), magrittr (>= 1.5), rlang (>= 1.0), stats, stringi (>= 1.7.8), utils
Suggests bench, covr, knitr, purrr, rmarkdown, spelling, testthat (>= 3.0)
VignetteBuilder knitr
Biarch TRUE
Config/testthat/edition 3
Encoding UTF-8
Language en-US
RoxygenNote 7.2.3
NeedsCompilation yes
Author Rory Nolan [aut, cre] (<https://orcid.org/0000-0002-5239-4043>)
Maintainer Rory Nolan <rorynolan@gmail.com>
Repository CRAN
Date/Publication 2023-10-30 04:50:02 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>before-and-after</td>
<td>2</td>
</tr>
<tr>
<td>currency</td>
<td>4</td>
</tr>
<tr>
<td>strex</td>
<td>5</td>
</tr>
<tr>
<td>str_alphord_nums</td>
<td>5</td>
</tr>
<tr>
<td>str_before_last_dot</td>
<td>6</td>
</tr>
<tr>
<td>str_can_be_numeric</td>
<td>7</td>
</tr>
<tr>
<td>str_detect_all</td>
<td>7</td>
</tr>
<tr>
<td>str_elem</td>
<td>8</td>
</tr>
<tr>
<td>str_elems</td>
<td>9</td>
</tr>
<tr>
<td>str_extract_non_numerics</td>
<td>10</td>
</tr>
<tr>
<td>str_extract_numbers</td>
<td>11</td>
</tr>
<tr>
<td>str_give_ext</td>
<td>13</td>
</tr>
<tr>
<td>str_locate_braces</td>
<td>13</td>
</tr>
<tr>
<td>str_locate_nth</td>
<td>14</td>
</tr>
<tr>
<td>str_match_arg</td>
<td>15</td>
</tr>
<tr>
<td>str_nth_non_numeric</td>
<td>17</td>
</tr>
<tr>
<td>str_nth_number</td>
<td>18</td>
</tr>
<tr>
<td>str_nth_number_after_mth</td>
<td>21</td>
</tr>
<tr>
<td>str_nth_number_before_mth</td>
<td>24</td>
</tr>
<tr>
<td>str_pasteelems</td>
<td>27</td>
</tr>
<tr>
<td>str_remove_quoted</td>
<td>28</td>
</tr>
<tr>
<td>str_singleize</td>
<td>29</td>
</tr>
<tr>
<td>str_split_by_numbers</td>
<td>30</td>
</tr>
<tr>
<td>str_split_camel_case</td>
<td>31</td>
</tr>
<tr>
<td>str_to_vec</td>
<td>31</td>
</tr>
<tr>
<td>str_trim_anything</td>
<td>32</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>before-and-after</td>
<td>34</td>
</tr>
</tbody>
</table>

**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**

```r
str_after_nth(string, pattern, n)
str_after_first(string, pattern)
str_after_last(string, pattern)
str_before_nth(string, pattern, n)
```
before-and-after

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::about_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")
### 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 data frame with 4 columns: `string_num`, `string`, `curr_sym` and `amount`. Every extracted currency amount gets its own row in the data frame 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.

**Usage**

```r
str_alphord_nums(string)
```

**Arguments**

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

A character vector.

Examples

```r
class(string) <- character
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

```r
str_before_last_dot(string)
```

Arguments

- `string`: A character vector.

Value

A character vector.

See Also

Other bisectors: `before-and-after`

Examples

```r
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**

```r
str_can_be_numeric(string)
```

**Arguments**

- `string` A character vector.

**Value**

A logical vector.

**Examples**

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

---

**str_detect_all**

Detect any or all patterns.

**Description**

Vectorized over `string`.

**Usage**

```r
str_detect_all(string, pattern, negate = FALSE)
str_detect_any(string, pattern, negate = FALSE)
```

**Arguments**

- `string` A character vector.
- `pattern` A character vector. The patterns to look for. Default is `stringr::coll()` and `stringr::fixed()` are also permissible.
- `negate` A flag. If TRUE, inverts the result.
Value

A character vector.

Examples

```r
genes <- c("MUC4", "MUC5B", "MUC6", "MUC7", "MUC8")
genome <- c("human", "mouse", "rat")
str_detect_all(genes, genome)  # TRUE
str_detect_any(genes, genome)  # TRUE
```

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: `str_elems()`, `str_pasteelems()`
**str_elems**

*Extract several single elements from a string.*

**Examples**

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

**Description**

Efficiently extract several elements from a string. See `str_elem()` for extracting single elements. This function is vectorized over the first argument.

**Usage**

```r
str_elems(string, indices, byrow = TRUE)
```

**Arguments**

- `string` A character vector.
- `indices` A vector of integerish values. Negative indexing is allowed as in `stringr::str_sub()`.
- `byrow` Should the elements be organised in the matrix with one row per string (`byrow = TRUE`, the default) or one column per string (`byrow = FALSE`). See examples if you don’t understand.

**Value**

A character matrix.

**See Also**

Other single element extractors: `str_elem()`, `str_paste_elems()`

**Examples**

```r
string <- c("abc", "def", "ghi", "vwxyz")
str_elems(string, 1:2)
str_elems(string, 1:2, byrow = FALSE)
str_elems(string, c(1, 2, 3, 4, -1))
```
str_extract_non_numerics

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

str_extract_non_numerics(
  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

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

See Also

Other non-numeric extractors: str_nth_non_numeric()
Examples

strings <- c(
  "abc123def456", "abc-0.12def.345", "abc.12e4def34.5e9",
  "abc1,100def1,230.5", "abc1,100e3,215def4e1,000"
)
str_extract_non_numerics(strings)
str_extract_non_numerics(strings, decimals = TRUE, leading_decimals = FALSE)
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).
Make the search aware of scientific notation e.g. 2e3 is the same as 2000.

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)).

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

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.

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.

Other numeric extractors: str_nth_number_after_mth(), str_nth_number_before_mth(), str_nth_number()

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_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
str_locate_nth

Value
A data frame 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

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

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::about_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**

```r
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**

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

```r
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
choices <- 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
```

str_nth_non_numeric

} 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).

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

Usage

\begin{verbatim}
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
)
\end{verbatim}
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.

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_non_numeric(strings, n = 2)
str_nth_non_numeric(strings, n = -2, decimals = TRUE)
str_first_non_numeric(strings, decimals = TRUE, leading_decimals = FALSE)
str_last_non_numeric(strings, commas = TRUE)
str_nth_non_numeric(strings,
  n = 1, decimals = TRUE, leading_decimals = TRUE,
  sci = TRUE
)
str_first_non_numeric(strings,
  decimals = TRUE, leading_decimals = TRUE,
  sci = TRUE, commas = TRUE, negs = TRUE
)
str_first_non_numeric(c("22", "1.2.3"), decimals = TRUE)
```
**str_nth_number**

**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**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A string.</td>
</tr>
<tr>
<td>n</td>
<td>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.</td>
</tr>
<tr>
<td>decimals</td>
<td>Do you want to include the possibility of decimal numbers (TRUE) or not (FALSE, the default).</td>
</tr>
<tr>
<td>leading_decimals</td>
<td>Do you want to allow a leading decimal point to be the start of a number?</td>
</tr>
<tr>
<td>negs</td>
<td>Do you want to allow negative numbers? Note that double negatives are not handled here (see the examples).</td>
</tr>
<tr>
<td>sci</td>
<td>Make the search aware of scientific notation e.g. 2e3 is the same as 2000.</td>
</tr>
</tbody>
</table>
str_nth_number

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 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**

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

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

str_nth_number_after_last(
    string, pattern, n,
    decimals = FALSE,
    leading_decimals = decimals,
    negs = FALSE,
    sci = FALSE,
    commas = FALSE,
    leave_as_string = FALSE
)
```
str_first_number_after_mth(
    string,
    pattern,
    m,
    decimals = FALSE,
    leading_decimals = decimals,
    negs = FALSE,
    sci = FALSE,
    commas = FALSE,
    leave_as_string = FALSE
)

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

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

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

str_last_number_after_first(

str_nth_number_after_mth

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

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

**Arguments**

- **string**  
  A character vector.

- **pattern**  
  The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_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.

- **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_before_mth()`, `str_nth_number()`

Examples

```r
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 n-th number before the m-th 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**

```r
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,
```
\texttt{str\_nth\_number\_before\_mth}

\begin{verbatim}
decimals = FALSE,
leading\_decimals = decimals,
negs = FALSE,
sci = FALSE,
commas = FALSE,
leave\_as\_string = FALSE
)
\end{verbatim}

\texttt{str\_nth\_number\_before\_last(}
\begin{verbatim}
string,
pattern,
n,
decimals = FALSE,
leading\_decimals = decimals,
negs = FALSE,
sci = FALSE,
commas = FALSE,
leave\_as\_string = FALSE
)
\end{verbatim}

\texttt{str\_first\_number\_before\_mth(}
\begin{verbatim}
string,
pattern,
m,
decimals = FALSE,
leading\_decimals = decimals,
negs = FALSE,
sci = FALSE,
commas = FALSE,
leave\_as\_string = FALSE
)
\end{verbatim}

\texttt{str\_last\_number\_before\_mth(}
\begin{verbatim}
string,
pattern,
m,
decimals = FALSE,
leading\_decimals = decimals,
negs = FALSE,
sci = FALSE,
commas = FALSE,
leave\_as\_string = FALSE
)
\end{verbatim}

\texttt{str\_first\_number\_before\_first(}
\begin{verbatim}
string,
pattern,
decimals = FALSE,
\end{verbatim}
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
)

Arguments

string A character vector.

pattern The pattern to look for.
The default interpretation is a regular expression, as described in stringi::about_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
str_paste elems

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: \texttt{str_extract_numbers()}, \texttt{str_nth_number_after_mth()}, \texttt{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_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 \texttt{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.
str_singleize

See Also

Other removers: str_singleize(), str_trim_anything()

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.
pattern The pattern to look for.
The default interpretation is a regular expression, as described in stringi::about_search_regex. To match a without regular expression (i.e. as a human would), use coll(). 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**

```r
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?
- `nugs`: 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)).

**Value**

A list of character vectors.

**See Also**

Other splitters: `str_split_camel_case()`

**Examples**

```r
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**


**See Also**

Other splitters: `str_split_by_numbers()`

**Examples**

```
str_split_camel_case(c("RoryNolan", "NaomiFlagg", "DepartmentOfSillyHats"))
str_split_camel_case(c("RoryNolan", "NaomiFlagg", "DepartmentOfSillyHats", lower = TRUE))
```

---

**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**

```
str_to_vec(string)
```
**str_trim_anything**

**Arguments**

- **string**: A character vector.

**Value**

A character vector.

**Examples**

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

---

**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 character vector.
- **pattern**: The pattern to look for. The default interpretation is a regular expression, as described in stringi::about_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

    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--", "--")
    str.Trim Anything("-ghi--", "i-+")
Index

* alphorderers
  str_alphord_nums, 5
* appenders
  str_give_ext, 13
* argument matchers
  str_match_arg, 15
* bisectors
  before-and-after, 2
  str_before_last_dot, 6
* converters
  str_to_vec, 31
* currency extractors
  currency, 4
* locators
  str_locate_braces, 13
  str_locate_nth, 14
* non-numeric extractors
  str_extract_non_numerics, 10
  str_nth_non_numeric, 17
* numeric extractors
  str_extract_numbers, 11
  str_nth_number, 18
  str_nth_number_after_mth, 21
  str_nth_number_before_mth, 24
* removers
  str_remove_quoted, 28
  str_singleize, 29
  str_trim_anything, 32
* single element extractors
  str.elem, 8
  str.elems, 9
  str_paste.elems, 27
* splitters
  str_split_by_numbers, 30
  str_split_camel_case, 31
* type converters
  str_can_be_numeric, 7

before-and-after, 2
coll(), 3, 14, 23, 26, 29, 32
currency, 4
match_arg (str_match_arg), 15
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, 6
str_before_nth (before-and-after), 2
str_can_be_numeric, 7
str_detect_all, 7
str_detect_any (str_detect_all), 7
str_elem, 8, 9, 28
str_elem(), 9
str.elems, 8, 9, 28
str.elems(), 27
str_extract_currencies (currency), 4
str_extract_currencies(), 4
str_extract_non_numerics, 10, 18
str_extract_numbers, 11, 20, 24, 27
str_extract_numbers(), 20
str_first_currency (currency), 4
str_first_non_numeric
  (str_nth_non_numeric), 17
str_first_number (str_nth_number), 18
str_first_number_after_first
  (str_nth_number_after_mth), 21
str_first_number_after_last
  (str_nth_number_after_mth), 21
str_first_number_after_mth
  (str_nth_number_after_mth), 21
str_first_number_before_first
  (str_nth_number_before_mth), 24
str_first_number_before_last
  (str_nth_number_before_mth), 24
INDEX

str_first_number_before_mth
  (str_nth_number_before_mth), 24
str_give_ext, 13
str_last_currency (currency), 4
str_last_non_numeric
  (str_nth_non_numeric), 17
str_last_number (str_nth_number), 18
str_last_number_after_first
  (str_nth_number_after_mth), 21
str_last_number_after_last
  (str_nth_number_after_mth), 21
str_last_number_after_mth
  (str_nth_number_after_mth), 21
str_last_number_before_first
  (str_nth_number_before_mth), 24
str_last_number_before_last
  (str_nth_number_before_mth), 24
str_last_number_before_mth
  (str_nth_number_before_mth), 24
str_locate_braces, 13, 15
str_locate_first (str_locate_nth), 14
str_locate_last (str_locate_nth), 14
str_locate_nth, 14, 14
str_match_arg, 15
str_nth_currency (currency), 4
str_nth_non_numeric, 10, 17
str_nth_number, 12, 18, 24, 27
str_nth_number_after_first
  (str_nth_number_after_mth), 21
str_nth_number_after_last
  (str_nth_number_after_mth), 21
str_nth_number_after_mth, 12, 20, 21, 27
str_nth_number_before_first
  (str_nth_number_before_mth), 24
str_nth_number_before_last
  (str_nth_number_before_mth), 24
str_nth_number_before_mth, 12, 20, 24, 24
str_paste_elems, 8, 9, 27
str_remove_quoted, 28, 29, 32
str_singleize, 29, 29, 32
str_split_by_numbers, 30, 31
str_split_camel_case, 30, 31
str_to_vec, 31
str_trim_anything, 29, 32
strex, 5
strex-package (strex), 5
stringi::about_search_regex, 3, 14, 23, 26, 29, 32
stringr::coll(), 7
stringr::fixed(), 7
stringr::regex(), 3, 14, 23, 26, 29, 32
stringr::str_sub(), 8, 9, 28