

Manage Lengths

Diagram illustrating a transformation or mapping process. The top row shows four colored squares (pink and light pink) arranged in a 2x4 grid. An arrow points down to a corresponding 2x4 grid of text labels: TRUE, FALSE, TRUE, TRUE.

Diagram illustrating a transformation or mapping process. The top row consists of four colored squares (pink and light pink). An arrow points down to a row of four text labels: TRUE, FALSE, TRUE, TRUE.

A diagram showing a 2x4 grid of squares. The top row has four squares, and the bottom row has four squares. The top row has a red square in the first position, an empty square in the second, a red square in the third, and a red square in the fourth. The bottom row has a red square in the first position, an empty square in the second, a red square in the third, and a red square in the fourth. Below the grid is a downward-pointing arrow. Below the arrow are the numbers 1, 2, and 4, each centered under a column of the grid above.

2 4
4 7
NANA
3 4

The diagram illustrates the merging of two sorted arrays. The top row shows two separate arrays: [1, 2, 3] and [4, 5, 6]. An arrow points down to the bottom row, which shows the merged array [1, 2, 3, 4, 5, 6].

ASTRING
a string

a string
↓
A STRING

a string
A String

4
1
3
2

Helpers

appl<e>
banana
p<e>ar

↓

TRU
TRU
FALL

This is a long sentence
This is a long sentence

Pattern arguments in stringr are interpreted as regular expressions *after any special characters have been parsed*.

In R, you write regular expressions as *strings*, sequences of characters surrounded by quotes ("" or single quotes(')).

Some characters cannot be represented directly in an R string. These must be represented as **special characters**, sequences of characters that have a specific meaning, e.g.

new line
/
"
/
"
/
"

Run `???` to see a complete list

Because of this, whenever a `\` appears in a regular expression, you must write it as `\\` in the string that represents the regular expression.

Use **writelines()** to see how R views your string after all special characters have been parsed.

```
writelines("\\ is a backslash")
# \\ is a backslash
```

Patterns in stringr are interpreted as regexes. To change this default, wrap the pattern in one of:

```
regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, doall = FALSE, ...)
```

lines as well of end of strings, allow R comments within regex's, and/or to have . match everything including \n.

```
str_detect("I", regex("i", TRUE))
```

fixed() Matches raw bytes but will miss some characters that can be represented in multiple ways (**fast**). `str_detect("u0130", fixed("ı"))`

coll() matches raw bytes and will use locale specific collation rules to recognize characters that can be represented in multiple ways (slow).
str_detect("u0130", coll("tr", TRUE, locale = "tr"))

boundary() Matches boundaries between characters, line_breaks, sentences, or words. `str_split(sentences, boundary("word"))`

Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

```
see <- function(rx) str_view_all("abcABC123\\t.!?\\\\{}\\n", rx)
```

¹ Many base R functions require classes to be wrapped in a second set of `[]`, e.g. `[[digit:]]`

```
alt<-function(rx) str_view_all("abcde", rx)
```

range alt("[a-c]") abcde

```
anchor<-function(rx) str_view_all("aaa", rx)
```

end of string anchor("\$a\$") aaa

```
look <- function(rx) str_view_all("bacad", rx)
```

not preceded by `look("(?<ib)a")` bacca

```
quant <- function(rx) str_view_all(".a.aa.aaa", rx)
```

```
a{n, m} between n and m quant("a{2,4}") .a.aa.aaa
```

```
ref <- function(rx) str_view_all("abbaab", rx)
```

```
first() group, etc.
ref("(a)(b)\\2\\1")
abbaab
```