The `stringr` package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

### Detect Matches

- `str_detect(string, pattern)` Detect the presence of a pattern match in a string. `str_detect(fruit, "a")`
- `str_which(string, pattern)` Find the indexes of strings that contain a pattern match. `str_which(fruit, "a")`
- `str_count(string, pattern)` Count the number of matches in a string. `str_count(fruit, "a")`
- `str_locate(string, pattern)` Locate the positions of pattern matches in a string. Also `str_locate_all(string, pattern)`
- `str_extract(string, pattern)` Return the first pattern match found in each string, as a vector. Also `str_extract_all(string, pattern)` to return every pattern match. `str_extract(fruit, "[aeiou"]`)
- `str_match(string, pattern)` Return the first pattern match found in each string, as a matrix with a column for each \( (\) group in pattern. Also `str_match_all(string, pattern)`

### Mutate Strings

- `str_sub(string, start = 1L, end = -1L)` Extract substrings from a character vector. `str_sub(fruit, 1, 3); str_sub(fruit, -2)`
- `str_subset(string, pattern)` Return only the strings that contain a pattern match. `str_subset(fruit, "b")`
- `str_replace(string, pattern, replacement)` Replace all matched patterns in each string. `str_replace(fruit, "a", "\"\")`
- `str_replace_all(string, pattern, replacement)` Replace all matched patterns in each string. `str_replace_all(fruit, "a", "\"\")`
- `str_to_title(string, locale = "en")` Convert strings to title case. `str_to_title(fruit, "en")`
- `str_to_lower(string, locale = "en")` Convert strings to lower case. `str_to_lower(fruit, "en")`
- `str_to_upper(string, locale = "en")` Convert strings to upper case. `str_to_upper(fruit, "en")`
- `str_split_fixed(string, pattern, \( n \)` Split a vector into a matrix of substrings (splitting at occurrences of a pattern match). Also `str_split_fixed(string, pattern, \( n \)`)
- `str_split(string, pattern)` Split a vector into substrings (splitting at occurrences of a pattern match). Also `str_split_all(string, pattern)`
- `str_length(string)` The width of strings (i.e. number of code points, which generally equals the number of characters). `str_length(fruit)`
- `str_pad(string, width, side = c("left", "right", "both"), pad = "")` Pad strings to constant width. `str_pad(fruit, 17)`
- `str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")` Truncate the width of strings, replacing content with ellipsis. `str_trunc(fruit, 3)`
- `str_trim(string, side = c("both", "left", "right"))` Trim whitespace from the start and/or end of a string. `str_trim(fruit)`

### Subset Strings

- `str_sub(string)` <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results. `str_sub(fruit, 1, 3) <- "str"`
- `str_to_title(sentences)` Convert strings to title case. `str_to_title(sentences)`
- `str_to_lower(sentences)` Convert strings to lower case. `str_to_lower(sentences)`
- `str_to_upper(sentences)` Convert strings to upper case. `str_to_upper(sentences)`
- `str_split_fixed(fruit, " ", n)` Split a vector of strings into a single string. `str_split_fixed(fruit, " ", n)`
- `str_split(fruit, " ")` Split strings into a single string. `str_split(fruit, " ")`
- `str_dup(string, times)` Repeat strings times. `str_dup(fruit, times = 2)`
- `str_pad(fruit, 17)` Pad strings to constant width. `str_pad(fruit, 17)`
- `str_length(fruit)` The width of strings (i.e. number of code points, which generally equals the number of characters). `str_length(fruit)`
- `str_pad(string, width, side = c("left", "right", "both"), pad = "")` Pad strings to constant width. `str_pad(fruit, 17)`
- `str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")` Truncate the width of strings, replacing content with ellipsis. `str_trunc(fruit, 3)`
- `str_trim(string, side = c("both", "left", "right"))` Trim whitespace from the start and/or end of a string. `str_trim(fruit)`

### Join and Split

- `str_c(..., sep = "", collapse = NULL)` Join multiple strings into a single string. `str_c(letters, LETTERS)`
- `str_c(..., sep = "", collapse = NULL)` Collapse a vector of strings into a single string. `str_c(letters, collapse = "")`
- `str_dup(string, times)` Repeat strings times. `str_dup(fruit, times = 2)`
- `str_pad(fruit, 17)` Pad strings to constant width. `str_pad(fruit, 17)`
- `str_length(fruit)` The width of strings (i.e. number of code points, which generally equals the number of characters). `str_length(fruit)`
- `str_pad(string, width, side = c("left", "right", "both"), pad = "")` Pad strings to constant width. `str_pad(fruit, 17)`
- `str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")` Truncate the width of strings, replacing content with ellipsis. `str_trunc(fruit, 3)`
- `str_trim(string, side = c("both", "left", "right"))` Trim whitespace from the start and/or end of a string. `str_trim(fruit)`

### Order Strings

- `str_sort(x)` Sort a character vector. `str_sort(fruit)`
- `str_sort(x)` Sort a character vector. `str_sort(fruit)`

### Manage Lengths

- `str_length(string)` The width of strings (i.e. number of code points, which generally equals the number of characters). `str_length(fruit)`
- `str_pad(string, width, side = c("left", "right", "both"), pad = "")` Pad strings to constant width. `str_pad(fruit, 17)`
- `str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")` Truncate the width of strings, replacing content with ellipsis. `str_trunc(fruit, 3)`
- `str_trim(string, side = c("both", "left", "right"))` Trim whitespace from the start and/or end of a string. `str_trim(fruit)`

### Helpers

- `str_conv(string, encoding)` Override the encoding of a string. `str_conv(fruit, ISO8859-1)`
- `str_view(string, pattern, match = NA)` View HTML rendering of first regex match in each string. `str_view(fruit, "[aeiou"]`)
- `str_view_all(string, pattern, match = NA)` View HTML rendering of all regex matches. `str_view_all(fruit, "[aeiou"]`)
- `str_wrap(string, width = 80, indent = 0, exdent = 0)` Wrap strings into nicely formatted paragraphs. `str_wrap(fruit)`

---

1 See bit.ly/ISO639-1 for a complete list of locales.
Need to Know

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.,

```
\n \n 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("\n #!
writeLines("\n is a backslash") # is a backslash
```

INTERPRETATION

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

**regex**{pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...}

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

```
str_detect("I", coll("i", TRUE, locale = "tr"))
```

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

**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("?", TRUE, locale = "tr"))`

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

MATCH CHARACTERS

<table>
<thead>
<tr>
<th>MATCH CHARACTERS</th>
<th>REGULAR EXPRESSIONS</th>
<th>MATCHES</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>string (type this)</td>
<td>regex (to mean this)</td>
<td>matches (which matches this)</td>
<td>example</td>
</tr>
</tbody>
</table>
| \a (etc.) | a (etc.) | see("a") | abc ABC 123 .!?

# \new line

\n
\n \n new line (return)

\t \t tab

\s \s any whitespace (\S for non-whitespace)

\w \w any word character (\W for non-word characters)

\b \b word boundaries

\1 \1 digits

[alpha] \[a-z]

[lower] \[a-z]

[upper] \[A-Z]

[alnum] \[a-zA-Z0-9]

[graph] \[0-9a-zA-Z\s!@#\%

[space] \[\s]

[blank] \[\n\r\t\n\r\t

\n \n 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("\n #!
writeLines("\n is a backslash") # is a backslash
```

INTERPRETATION

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

**regex**{pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...}

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

```
str_detect("I", coll("i", TRUE, locale = "tr"))
```

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

**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("?", TRUE, locale = "tr"))`

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