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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_detect()</code></td>
<td>Detect the presence of a pattern match in a string.</td>
</tr>
<tr>
<td><code>str_match()</code></td>
<td>Return every pattern match found in each string, as a matrix with a column for each () group in pattern.</td>
</tr>
<tr>
<td><code>str_locate_all()</code></td>
<td>Find the indexes of strings that contain a pattern match.</td>
</tr>
<tr>
<td><code>str_locate()</code></td>
<td>Return the first pattern match found in each string, as a vector.</td>
</tr>
</tbody>
</table>

### Subset Strings

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_sub()</code></td>
<td>Extract substrings from a character vector.</td>
</tr>
<tr>
<td><code>str_subset()</code></td>
<td>Return only the strings that contain a pattern match.</td>
</tr>
<tr>
<td><code>str_extract()</code></td>
<td>Return the first pattern match found in each string, as a vector.</td>
</tr>
<tr>
<td><code>str_match()</code></td>
<td>Return the first pattern match found in each string, as a matrix with a column for each () group in pattern.</td>
</tr>
</tbody>
</table>

### Mutate Strings

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_c()</code></td>
<td>Join multiple strings into a single string.</td>
</tr>
<tr>
<td><code>str_replace()</code></td>
<td>Replace a vector of strings by identifying the substrings with <code>str_sub()</code> and assigning into the results.</td>
</tr>
<tr>
<td><code>str_replace_all()</code></td>
<td>Replace all matched patterns in each string.</td>
</tr>
<tr>
<td><code>str_to_lower()</code></td>
<td>Convert strings to lowercase.</td>
</tr>
<tr>
<td><code>str_to_upper()</code></td>
<td>Convert strings to uppercase.</td>
</tr>
<tr>
<td><code>str_to_title()</code></td>
<td>Convert strings to title case.</td>
</tr>
</tbody>
</table>

### Join and Split

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_collapse()</code></td>
<td>Collapse strings with a character or string.</td>
</tr>
<tr>
<td><code>str_split_fixed()</code></td>
<td>Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match).</td>
</tr>
<tr>
<td><code>str_split()</code></td>
<td>Split strings into a vector of substrings.</td>
</tr>
<tr>
<td><code>str_wrap()</code></td>
<td>Trim whitespace from the start and/or end of a string.</td>
</tr>
</tbody>
</table>

### Manage Lengths

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_length()</code></td>
<td>The width of strings (i.e. number of code points, which generally equals the number of characters).</td>
</tr>
<tr>
<td><code>str_pad()</code></td>
<td>Pad strings to constant width.</td>
</tr>
<tr>
<td><code>str_trunc()</code></td>
<td>Truncate the width of strings, replacing content with ellipsis.</td>
</tr>
<tr>
<td><code>str_which()</code></td>
<td>Return a list of substrings.</td>
</tr>
</tbody>
</table>

### Order Strings

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_order()</code></td>
<td>Sort a vector of indexes that sorts a character vector.</td>
</tr>
<tr>
<td><code>str_sort()</code></td>
<td>Sort a character vector.</td>
</tr>
<tr>
<td><code>str_order()</code></td>
<td>Sort a list of substrings.</td>
</tr>
</tbody>
</table>

### Helpers

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str_conv()</code></td>
<td>Override the encoding of a string.</td>
</tr>
<tr>
<td><code>str_view()</code></td>
<td>View HTML rendering of a string.</td>
</tr>
<tr>
<td><code>str_glue_data()</code></td>
<td>Create a string from expressions to evaluate.</td>
</tr>
</tbody>
</table>

---

### Need to Know

#### Pattern Arguments

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.

- Special Character: `\` Represents
  - `\n` new line
  - `\r` new line
  - `\t` tab
  - `\b` word boundaries
- `\w` any word character
- `\s` any whitespace
- `\t` tab
- `\n` new line (return)
- `\{` repetition
- `\}` repetition

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

```r
writeLines("\"\")
writeLines("\" is a backslash\"")
```

#### INTERPRETATION

Patterns in stringr are interpreted as regular expressions, or **regexps**, a concise language for describing patterns in strings. `\` is a backslash and `\{m\}` matches raw bytes but will miss some sequences of characters surrounded by quotes (e.g. `"abc"`) or single quotes (e.g. `'abc'`).

Use parentheses to set precedent (order of evaluation) and create groups:

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

### MATCH CHARACTERS

Patterns in stringr are interpreted as regular expressions, or **regexps**, a concise language for describing patterns in strings. `\` is a backslash and `\{m\}` matches raw bytes but will miss some sequences of characters surrounded by quotes (e.g. `"abc"`) or single quotes (e.g. `'abc'`).

<table>
<thead>
<tr>
<th><strong>MATCH CHARACTERS</strong></th>
<th>Matches</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\w</code></td>
<td>any word character</td>
<td><code>ab</code></td>
</tr>
<tr>
<td><code>\s</code></td>
<td>any whitespace</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\t</code></td>
<td>tab</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\n</code></td>
<td>new line (return)</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\{</code></td>
<td>repetition</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\}</code></td>
<td>repetition</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\()</code></td>
<td>( can only appear in a pattern.</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\)</code></td>
<td>) can only appear in a pattern.</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\{n,}</code></td>
<td>one or more</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\{0,}</code></td>
<td>zero or more</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\{1\}</code></td>
<td>one</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\{n\}</code></td>
<td>exactly n</td>
<td><code>abc ABC 123</code></td>
</tr>
<tr>
<td><code>\{n,\}</code></td>
<td>n or more</td>
<td><code>abc ABC 123</code></td>
</tr>
</tbody>
</table>

### QUANTIFIERS

Use parentheses to set precedent (order of evaluation) and create groups:

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

### LOOK AROUNDS

Use an escaped number to refer to and duplicate parentheses groups that occur earlier in a pattern. Refer to each group by its order of appearance:

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