String manipulation with stringr:: CHEAT SHEET

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, negate = FALSE) Detect the presence of a pattern match in a string. Also str like(). str detect(fruit, "a")

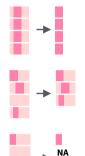
str_starts(string, pattern, negate = FALSE) Detect the presence of a pattern match at the beginning of a string. Also **str_ends()**. str starts(fruit, "a")

str_which(string, pattern, negate = FALSE) Find the indexes of strings that contain a pattern match. str which(fruit, "a")

str_locate(string, pattern) Locate the positions of pattern matches in a string. Also str locate all(). str locate(fruit, "a")

str_count(string, pattern) Count the number of matches in a string. str count(fruit, "a")

Subset 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, negate = FALSE) Return only the strings that contain a pattern match. str_subset(fruit, "p")

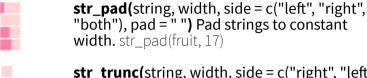
str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also **str_extract_all()** to return every pattern match. str extract(fruit, "[aeiou]")

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(). str match(sentences, "(althe) ([^ +])")

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_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. str trunc(sentences, 6)



str_squish(string) Trim whitespace from each end and collapse multiple spaces into single spaces. str squish(str pad(fruit, 17, "both"))

Mutate Strings



str sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results.

str sub(fruit, 1, 3) <- "str"

str replace(string, **pattern**, replacement) Replace the first matched pattern in each string. Also str_remove().

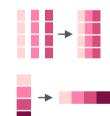
str_replace(fruit, "p", "-")

str_replace_all(string, pattern, replacement) Replace all matched patterns in each string. Also **str_remove_all()**.

str replace all(fruit, "p", "-")

strings to title case. Also **str_to_sentence()**. str_to_title(sentences)

Join and Split



{xx} {yy}

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. str c(letters, LETTERS)

str flatten(string, collapse = "") Combines into a single string, separated by collapse. str flatten(fruit, ", ")

str_dup(string, times) Repeat strings times times. Also **str_unique()** to remove duplicates. str dup(fruit, times = 2)

str_split_fixed(string, **pattern**, n**)** Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split()** to return a list of substrings and **str_split_n()** to return the nth substring. str split fixed(sentences, " ", n=3)

str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. str_glue("Pi is {pi}")

parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")

str_glue_data(.x, ..., .sep = "", .envir =

Order Strings



str order(x, decreasing = FALSE, na last = TRUE, locale = "en", numeric = FÁLSE, ...)1 Return the vector of indexes that sorts a character vector. fruit[str order(fruit)]



str sort(x, decreasing = FALSE, na last = TRUE, locale = "en", numeric = FALSE, ...)1 **Sort a character vector.** str_sort(fruit)

Helpers

appl<e> banana p<e>ar

TRUE

FALSE

This is a long sentence. **▼** This is a long

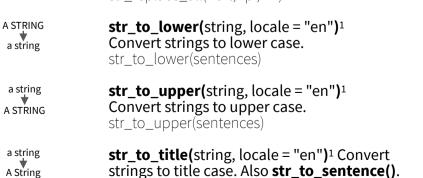
str_conv(string, encoding) Override the encoding of a string. str_conv(fruit,"ISO-8859-1")

str_view_all(string, pattern, match = NA) View HTML rendering of all regex matches. Also **str_view()** to see only the first match. str_view_all(sentences, "[aeiou]")

str_equal(x, y, locale = "en", ignore_case = FALSE, ...)¹ Determine if two strings are equivalent. str_equal(c("a", "b"), c("a", "c"))

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. str_wrap(sentences, 20)

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.

Special Character	Represents
//	\
\"	II .
\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("\\.")
writeLines("\\ is a backslash")
#\is a backslash
```

INTERPRETATION

Patterns in stringr are interpreted as regexs. 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 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("i"))

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("i", TRUE, locale = "tr"))

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

[:graph:]

[:space:]

[:blank:]

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

		9 Part 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	801		
MATCH CHARACTERS		see <- function(rx) str_view_all("abc ABC 123\t.!?\\(){}\n", rx)			
string (type this)	regexp (to mean this)	matches (which matches this)	example		
	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}	
\\.	\.		see("\\.")	abc ABC 123 .!?\(){}	
\\!	\!	!	see("\\!")	abc ABC 123 .!?\(){}	
\\?	\?	?	see("\\?")	abc ABC 123 .! <mark>?</mark> \(){}	
\\\\	\\		see("\\\\")	abc ABC 123 .!? <mark>\</mark> (){}	
\\(\((see("\\(")	abc ABC 123 .!?\ <mark>(</mark>){}	
\\)	\))	see("\\)")	abc ABC 123 .!?\(<mark>)</mark> {}	
\\ {	\{	{	see("\\{")	abc ABC 123 .!?\() <mark>{</mark> }	
\\ }	\}	}	see("\\}")	abc ABC 123 .!?\(){ <mark>}</mark>	
\\ n	\n	new line (return)	see("\\n")	abc ABC 123 .!?\(){}	
\\t	\t	tab	see("\\t")	abc ABC 123 .!?\(){}	
\\s	\s	any whitespace (\ S for non-whitespaces)	see("\\s")	abc ABC 123 .!?\(){}	
\\ d	\ d	any digit (\ D for non-digits)	see("\\d")	abc ABC <mark>123</mark> .!?\(){}	
\\w	\w	any word character (\W for non-word chars)	see("\\w")	abc ABC 123 .!?\(){}	
\\ b	\ b	word boundaries	see("\\b")	abc ABC 123 .!?\(){}	
	[:digit:]	digits	see("[:digit:]")	abc ABC <mark>123</mark> .!?\(){}	
	[:alpha:]	letters	see("[:alpha:]")	abc ABC 123 .!?\(){}	
	[:lower:]	lowercase letters	see("[:lower:]")	abc ABC 123 .!?\(){}	
	[:upper:]	uppercase letters	see("[:upper:]")	abc <mark>ABC</mark> 123 .!?\(){}	
	[:alnum:]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}	
[:punct:] 1		punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}	



see("[:graph:]")

see("[:space:]")

see("[:blank:]")

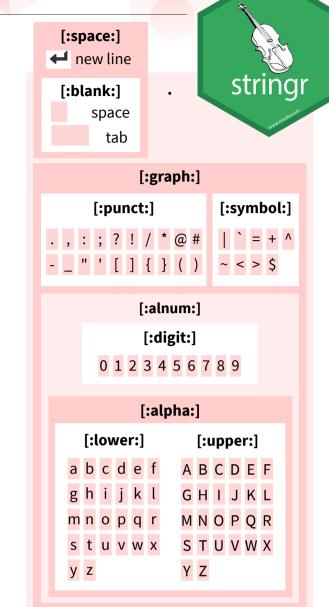
see(".")

abc ABC 123 .!?\(){}

abc ABC 123 .!?\(){}

abc ABC 123 .!?\(){}

abc ABC 123 .!?\(){}



ALTERNATES	alt <- function(rx) str_view_all("abcde", rx)			
	regexp	matches	example	
	ab d	or	alt("ab d")	abcde
	[abe]	one of	alt("[abe]")	abcde
	[^abe]	anything but	alt("[^abe]")	abcde
	[a-c]	range	alt("[a-c]")	abcde
ANCHORS		anchor <- func	tion(rx) str_view_al	l("aaa", rx)
	regexp	matches	example	
	^a	start of string	anchor("^a")	aaa
	a \$	end of string	anchor("a\$")	aaa

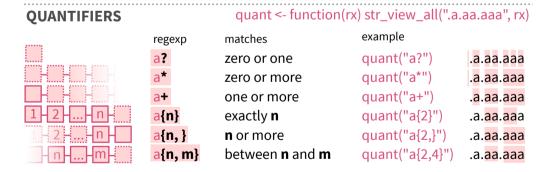
letters, numbers, and punctuation

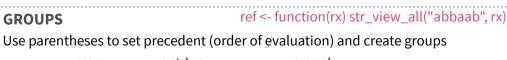
space and tab (but not new line)

every character except a new line

space characters (i.e. \s)

LOOK AROUNDS	look <- function(rx) str_view_all("bacad", rx)			
	regexp	matches	example	
	a(?=c)	followed by	look("a(?=c)")	b <mark>a</mark> cad
	a(?!c)	not followed by	look("a(?!c)")	bac <mark>a</mark> d
	(?<=b)a	preceded by	look("(?<=b)a")	b <mark>a</mark> cad
	(? b)a</th <th>not preceded by</th> <th>look("(?<!--b)a")</th--><th>bac<mark>a</mark>d</th></th>	not preceded by	look("(? b)a")</th <th>bac<mark>a</mark>d</th>	bac <mark>a</mark> d





matches example regexp (ab|d)e alt("(ab|d)e") abcde sets precedence

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

string	regexp	matches	example (the result is the same as ref("abba"))
(type this)	(to mean this)	(which matches this)	
\\1	\1 (etc.)	first () group, etc.	$ref("(a)(b)\\2\1")$ abbaab