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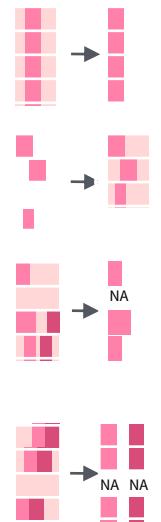
The stringr package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

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, "(a|the) ([^ +])")

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(sentences, 6)

str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. str_trim(str_pad(fruit, 17))

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", "-")

str_to_lower(string, locale = "en")[1] Convert strings to lower case. str_to_lower(sentences)

str_to_upper(string, locale = "en")[1] Convert strings to upper case. str_to_upper(sentences)

str_to_title(string, locale = "en")[1] Convert strings to title case. Also str_to_sentence(). str_to_title(sentences)

Join and Split

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_i() to return the ith 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}")

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

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, …)[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

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

appl\<e\>
banana
p\<e\>ar

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"))

This is a long sentence. This is a long sentence.

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
\\	\
\"	"
\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"))

Regular Expressions -
Regular expressions, or *regexps*, are a concise language for describing patterns in strings.
see <- function(rx) str_view_all("abc ABC 123\t.!?\\(){}\n", rx)

MATCH CHARACTERS

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 .!?\(){}
\\\\	\\	\	see("\\\\")	abc ABC 123 .!?\(){}
\\(	\(	(	see("\\(")	abc ABC 123 .!?\(){}
\\)	\)	)	see("\\)")	abc ABC 123 .!?\(){}
\\{	\{	{	see("\\{")	abc ABC 123 .!?\(){}
\\}	\}	}	see ("\\}")	abc ABC 123 .!?\(){}
\\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 123 .!?\(){}
\\w	\w	any word character (\W *for non-word chars*)	see("\\w")	abc ABC 123 .!?\(){}
\\b	\b [1]	word boundaries	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:]	digits	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:]	letters	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:]	lowercase letters	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:]	uppercase letters	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:]	punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:]	letters, numbers, and punctuation	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:]	space characters (i.e. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:]	space and tab (but not new line)	see("[:blank:]")	abc ABC 123 .!?\(){}
	.	every character except a new line	see(".")	abc ABC 123 .!?\(){}

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

[:space:]
↵ new line

[:blank:]
space
tab

[:graph:]

[:punct:]				[:symbol:]			
.	,	:	;		|	`	=	+	^
?	!	/	*	@	#					
-	_	"		[ ] { } ( )		~	<	>	$

[:alnum:]

[:digit:]
0 1 2 3 4 5 6 7 8 9

[:alpha:]

[:lower:]	[:upper:]
a b c d e f	A B C D E F
g h i j k l	G H I J K L
m n o p q r	M N O P Q R
s t u v w x	S T U V W X
y z	Y Z

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 <- function(rx) str_view_all("aaa", rx)
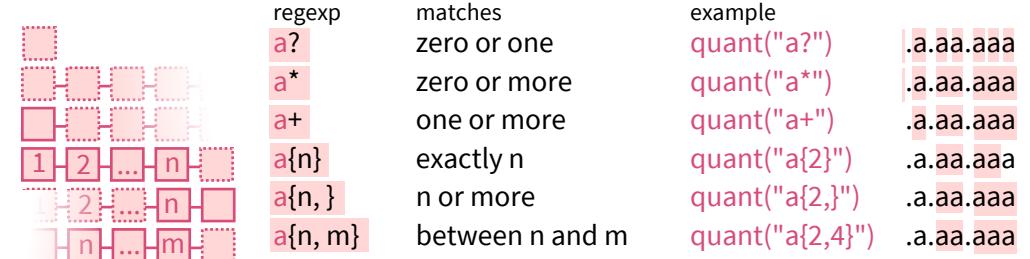
regexp	matches	example	
^a	start of string	anchor("^a")	aaa
a$	end of string	anchor("a$")	aaa

LOOK AROUNDS
look <- function(rx) str_view_all("bacad", rx)

regexp	matches	example	
a(?=c)	followed by	look("a(?=c)")	bacad
a(?!c)	not followed by	look("a(?!c)")	bacad
(?<=b)a	preceded by	look("(?<=b)a")	bacad
(?<!b)a	not preceded by	look("(?<!b)a")	bacad

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

regexp	matches	example	
a?	zero or one	quant("a?")	.a.aa.aaa
a*	zero or more	quant("a*")	.a.aa.aaa
a+	one or more	quant("a+")	.a.aa.aaa
a{n}	exactly n	quant("a{2}")	.a.aa.aaa
a{n, }	n or more	quant("a{2,}")	.a.aa.aaa
a{n, m}	between n and m	quant("a{2,4}")	.a.aa.aaa

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

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

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

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