## 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("").

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

Special Character Represents Run ?"" to see a complete list new line

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 str\_detect("I", regex("i", TRUE)) including

fixed() Matches raw bytes but will miss some characters that can be represented in multiple ways (fast). str\_detect("\u00e40130", fixed("i"))

specific collation rules to recognize characters that can be represented in multiple ways (slow).  $str\_detect("|u0130", coll("i", TRUE, locale="tr"))$ coll() Matches raw bytes and will use locale

characters, line\_breaks, sentences, or words. str\_split(sentences, boundary("word")) boundary() Matches boundaries between

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

see("\")"  see("\")"  see("\"\")  see("\"\")  see("\"\\")  see("\\\")  see("\\\\")  see("\\\\")  see("\\\\\")  see("\\\\\\")  see("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	MATCH C	MATCH CHARACTERS	see <- function(rx)	str_view_all("abc A	$see <-function(rx) str\_view\_all("abcABC 123\t.!?\\() \{ \\n", rx \)$
a (etc.)   a (etc.)     .   .   .   .   .   .   .   .	<b>string</b> (type this)	regexp (to mean this)	matches (which matches this)	example	
		a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
	<u>:</u>			see("\\")	abc ABC 123 [!?\(){}
	<b>:</b>	11		see("\\!")	abc ABC 123!?\(){}
	113	13	5	see("\\?")	abc ABC 123 .!?\(){}
(				see("\\\\")	abc ABC 123 .!?\(){}
	<u> </u>	//		see("/\(")	abc ABC 123 .!?\()\{}
{   See("\\")       See("\\")         See("\\")         See("\\")       See("\\")       See("\\")     See("\\")     See("\\")     See("\\")     See("\\")     See("\\")     See("\\")     See("\\")     See("\\")     See("\\")     See("\\")     See("\\")   See("\\")     See(\\")     See(\\")     See(\\")     See(\\")     See(\\")     See(\\")     See(\\")   See(\\")     See(\\")     See(\\")   See(\\")   See(\\")     See(\\")	<b>=</b>	2		see("//)")	abc ABC 123 .!?\(0){}
	<b>}</b>	M	_	see("/\{")	abc ABC 123 .!?\(){}
n new line (return)   see("\n")   tab   any whitespace (\structure{S} for non-whitespaces)   see("\n")   see(\n")   see(\	<b>=</b>	7		see( "\\\]")	abc ABC 123 .!?\(){}
tab   see("\ \tr\")     any whitespace (\S for non-whitespaces)     any whitespace (\S for non-whitespaces)     any word character (\M for non-word chars)     any word boundaries   see("\ \hat{v}")     cligit:   digits   see("\ \hat{v}")     cligit:   letters   see("\ \hat{v}")     clific:   see("\ \hat{v}")     clific:	//u	n/	new line (return)	see("\\n")	abc ABC 123 .!?\(){}
See("\ \s")   See("\ \s")     A any whitespace (\stractions for non-whitespaces)   See("\ \d")     A any digit (\textstyle{\textsty}}}}}}}}}}}}}}}}}}}}}    O not not now line   Den not not now lin	//t	/ <del>t</del>	tab	see("\\t")	abc ABC 123 .!?\(){}
d any digit \( \D \text{for non-digits} \) see("\ \d") \\ word boundaries \\ \text{see("\ \d")} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\	s	18	any whitespace (\ <b>S</b> for non-whitespaces)	see("\\s")	abc ABC 123 .!?\(){}
w any word character (\text{W for non-word chars}) see("\ \w"\ ) word boundaries    cidigit:     cidigit:     cidigit:     cidigit:     cidigit:     cidigit:     cidigit:     cidigit:     cidigit:	p//	<b>P</b> /	any digit ( <b>\D</b> for non-digits)	see("/\d")	
word boundaries   see("\ \bar{\text{"}}\ \bar{\text{"}}\ \end{alignments}   digits   see("\ \cdot\	//w	/w	any word character (\W for non-word chars)	see("\\w")	
digits letters letters lowercase letters uppercase letters uppercase letters letters and numbers letters and numbers letters, numbers, and punctuation space characters (i.e. \s) space and tab (but not new line) letters and tab (but not new line) see("[:blank:]") space and tab (but not new line) see("[:blank:]") see("[:blank:]") see("[:blank:]")	q//	q\	word boundaries	see("\\b")	abc ABC 123 .!?\(){}
letters lowercase letters lowercase letters uppercase letters letters and numbers letters, numbers, and punctuation space characters (i.e. \s) space and tab (but not new line) see("[:space:]") space and tab (but not new line) see("[:space:]") space and tab (but not new line) see("[:space:]") see("[:space:]")		[:digit:]	digits	see("[:digit:]")	
lowercase letters see("[:lower:]") uppercase letters see("[:upper:]") letters and numbers see("[:alnum:]") punctuation see("[:alnum:]") letters, numbers, and punctuation see("[:graph:]") space characters (i.e. \s) see("[:space:]") space and tab (but not new line) see("[:blank:]") every character except a new line see(":")		[:albha:]	letters	see("[:alpha:]")	
uppercase letters see("[:upper:]") letters and numbers see("[:alnum:]") punctuation see("[:punct:]") letters, numbers, and punctuation see("[:graph:]") space characters (i.e. \s) see("[:space:]") space and tab (but not new line) see("[:blank:]") every character except a new line see(":)		[:lower:]	lowercase letters	see("[:lower:]")	
letters and numbers see("[:alnum:]")  punctuation see("[:punct:]")  letters, numbers, and punctuation see("[:graph:]")  space characters (i.e. \s) see("[:space:]")  space and tab (but not new line) see("[:blank:]")  every character except a new line see(":)		[:nbber:]	uppercase letters	see("[:nbber:]")	123
punctuation see("[;punct:]") letters, numbers, and punctuation see("[;graph:]") space characters (i.e. \s) see("[;space:]") see("[;blank:]") every character except a new line see("")		[:alunm:]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}
letters, numbers, and punctuation see("[ˈgraph:]") space characters (i.e. \s) see("[:space:]") see("[:blank:]") every character except a new line see(":")		[:punct:]	punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}
space characters (i.e. \s) space and tab (but not new line) space and tab (but not new line) space and tab (but not new line) space (":blank:]")		[:graph:]	letters, numbers, and punctuation	see("[:graph:]")	abc ABC 123 .!?\(){}
space and tab (but not new line)  every character except a new line  see("":blank:]")		[:space:]	space characters (i.e. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
see("")		[:blank:]	space and tab (but not new line)	see("[:blank:]")	abc ABC 123 .!?\(){}
			every character except a new line	see(":")	abc ABC 123 .!?\(){}

•

[:alnnm:]

[:graph:] [:punct:]

space

**→** new line [:blank:] tab

				ш	_	~	~		
					~				
	9		[:upper:]						
	3 9		ğ		7				
	7 8		늄		-				
_	-	_	ت	В	工				
∓	9	ä		A	G	Σ	S	Z	
[:digit:]	4 5	[:alpha:]							
3	3 4	<u> </u>		Ŧ	_	_	×		
	7		$\overline{\cdot}$	е	$\prec$	d	>		
	1 2		[:lower:]	р	-	р	>		
	0		8	C		0	n		
			≕	þ	7	$\Box$	+		
				В	<b>₽</b> 0	Ε	S	Z	

ALTERNATES		alt <- function	alt <- function(rx) str_view_all("abcde", rx)	abcde", rx)	QUANTIFIERS	3S	quant	<- function(r	quant <- function(rx) str_view_all(".a.aa.aaa", r>	aa.aaa", r
	regexp	matches	example			regexp	xp matches		example	
	ab d	or	alt("ab d")	abcde		a?	zero or one	ne	quant("a?")	.a.aa.aa
	abe	one of	alt("[abe]")	abcde		a*	zero or more	nore	quant("a*")	.a.aa.aag
	[^abe]	anything but	alt("[^abe]")	abcde		a+	one or more	ore	quant("a+")	.a.aa.aag
	a-c	range	alt("[a-c]")	abcde	1-2n-	a{ <b>n</b> }	exactly <b>n</b>	_	quant("a{2}")	.a.aa.aag
					1 - 2	H a{ <b>n</b> , }	) n or more	ė	quant("a{2,}")	.a.aa.aaa
ANCHORS		anchor <- function	anchor <- function(rx) str_view_all("aaa", rx)	("aaa", rx)	- n m	10	{ <b>n, m</b> } between	between <b>n</b> and <b>m</b>	quant("a{2,4}")	.a.aa.aas
	regexp	matches	example							
	۸a	start of string	anchor("^a")	aaa	GROUPS		re	function	ref <- function(rx) str_view_all("abbaab", rx	bbaab", n
	<b>a\$</b>	end of string	anchor("a\$")	aaa	Use parenthe	ses to set p	recedent (order	of evaluatior	Use parentheses to set precedent (order of evaluation) and create groups	S
					re	regexp	matches		ole	
LOOK AROUNDS		look <- function(	look <- function(rx) str_view_all("bacad", rx)	acad", rx)	(a	(ab d)e	sets precedence		alt("(ab d)e")	apcde
	regexp	matches	example		Use an escape	ed number	to refer to and du	uplicate pare	Use an escaped number to refer to and duplicate parentheses groups that occur	at occur
	a(?=c)	followed by	look("a(?=c)")	bacad	earlier in a pa	ttern. Refer	earlier in a pattern. Refer to each group by its order of appearance	y its order of	appearance	
X	a(?!c)	not followed by	look("a(?!c)")	bacad		regexp	matches		lle	( P.
.) :):::	(?<=b)a	preceded by	look("(?<=b)a")	bacad	(type tnis) (to	(to mean this)	(wnich matches this)		(tne result is tne same as rer( "abba"))	appa ))
X	(3 <ib)a< th=""><th>not preceded by</th><th>look("(?<!--b)a")</th--><th>bacad</th><th>//1 //1</th><th><b>1</b> (etc.)</th><th>first () group, etc.</th><th></th><th>ref("(a)(b)\\2\\1")</th><th>abbaab</th></th></ib)a<>	not preceded by	look("(? b)a")</th <th>bacad</th> <th>//1 //1</th> <th><b>1</b> (etc.)</th> <th>first () group, etc.</th> <th></th> <th>ref("(a)(b)\\2\\1")</th> <th>abbaab</th>	bacad	//1 //1	<b>1</b> (etc.)	first () group, etc.		ref("(a)(b)\\2\\1")	abbaab

.a.aa.aaa

