Package 'rex'

December 5, 2016

Type Package

Title Friendly Regular Expressions

Version 1.1.1

Maintainer Jim Hester <james.f.hester@gmail.com>

URL https://github.com/kevinushey/rex

BugReports https://github.com/kevinushey/rex/issues

Description A friendly interface for the construction of regular expressions.

Imports magrittr, lazyeval

License MIT + file LICENSE

Suggests testthat, knitr, rmarkdown, dplyr, ggplot2, lintr, Hmisc, stringr, rvest, roxygen2

VignetteBuilder knitr

Collate 'aaa.R' 'utils.R' 'escape.R' 'capture.R' 'character_class.R' 'counts.R' 'lookarounds.R' 'match.R' 'or.R' 'rex-mode.R' 'rex.R' 'shortcuts.R' 'wildcards.R' 'zzz.R'

RoxygenNote 5.0.1.9000

NeedsCompilation no

Author Kevin Ushey [aut], Jim Hester [aut, cre], Robert Krzyzanowski [aut]

Repository CRAN

Date/Publication 2016-12-05 18:28:48

R topics documented:

`as.character.regex` *Regular Expression*

Description

Specify an explicit regular expression. This expression must already be escaped.

Usage

```
## S3 method for class 'regex'
as.character(x, ...)

## S3 method for class 'regex'
print(x, ...)

regex(x, ...)
```

Arguments

`x`	Object
`...`	further arguments

Methods (by generic)

- `as.character`: coerce regex object to a character
- `print`: Print regex object

See Also

`as.regex` to coerce to a regex object.

as.regex *Coerce objects to a* regex.

Description

Coerce objects to a regex.

Usage

```
as.regex(x, ...)

## Default S3 method:
as.regex(x, ...)
```

Arguments

x	Object to coerce to regex.
...	further arguments passed to methods.

Methods (by class)

- default: Simply escape the Object.

capture *Create a capture group*

Description

Used to save the matched value within the group for use later in the regular expression or to extract the values captured. Both named and unnamed groups can later be referenced using capture_group.

Usage

```
capture(..., name = NULL)

capture_group(name)
```

Arguments

...	shortcuts, R variables, text, or other **rex** functions.
name	of the group. Unnamed capture groups are numbers starting at 1 in the order they appear in the regular expression. If two groups have the same name, the leftmost group is the used in any reference.

See Also

group for grouping without capturing. Perl 5 Capture Groups http://perldoc.perl.org/perlre.html#Capture-groups

Other rex: %or%, character_class, counts, group, lookarounds, not, rex, shortcuts, wildcards

Examples

```
# Match paired quotation marks
re <- rex(
  # first quotation mark
  capture(quotes),

  # match all non-matching quotation marks
  zero_or_more(except(capture_group(1))),

  # end quotation mark (matches first)
  capture_group(1)
)

#named capture - don't match apples to oranges
re <- rex(
  capture(name = "fruit", or("apple", "orange")),
  "=",
  capture_group("fruit")
)
```

---

character_class *Create character classes*

Description

There are multiple ways you can define a character class.

Usage

```
character_class(x)

one_of(...)

any_of(..., type = c("greedy", "lazy", "possessive"))

some_of(..., type = c("greedy", "lazy", "possessive"))

none_of(...)

except_any_of(..., type = c("greedy", "lazy", "possessive"))
```

```
except_some_of(..., type = c("greedy", "lazy", "possessive"))

range(start, end)

":"(start, end)

exclude_range(start, end)
```

Arguments

x	text to include in the character class (must be escaped manually)
...	shortcuts, R variables, text, or other **rex** functions.
type	the type of match to perform. There are three match types 1. greedy: match the longest string. This is the default matching type. 2. lazy: match the shortest string. This matches the shortest string from the same anchor point, not necessarily the shortest global string. 3. possessive: match and don't allow backtracking
start	beginning of character class
end	end of character class

Functions

- character_class: explicitly define a character class
- one_of: matches one of the specified characters.
- any_of: matches zero or more of the specified characters.
- some_of: matches one or more of the specified characters.
- none_of: matches anything but one of the specified characters.
- except_any_of: matches zero or more of anything but the specified characters.
- except_some_of: matches one or more of anything but the specified characters.
- range: matches one of any of the characters in the range.
- :: matches one of any of the characters in the range.
- exclude_range: matches one of any of the characters except those in the range.

See Also

Other rex: %or%, capture, counts, group, lookarounds, not, rex, shortcuts, wildcards

Examples

```
# grey = gray
re <- rex("gr", one_of("a", "e"), "y")
grepl(re, c("grey", "gray")) # TRUE TRUE

# Match non-vowels
```

```
re <- rex(none_of("a", "e", "i", "o", "u"))
# They can also be in the same string
re <- rex(none_of("aeiou"))
grepl(re, c("k", "l", "e")) # TRUE TRUE FALSE

# Match range
re <- rex(range("a", "e"))
grepl(re, c("b", "d", "f")) # TRUE TRUE FALSE

# Explicit creation
re <- rex(character_class("abcd\\["))
grepl(re, c("a", "d", "[", "]")) # TRUE TRUE TRUE FALSE
```

`character_class_escape` *Character class escapes*

Description

Character class escapes

Usage

```
character_class_escape(x)

## S3 method for class 'regex'
character_class_escape(x)

## S3 method for class 'character_class'
character_class_escape(x)

## S3 method for class 'character'
character_class_escape(x)

## S3 method for class 'list'
character_class_escape(x)

## Default S3 method:
character_class_escape(x)
```

Arguments

x	Object to escape.

Methods (by class)

- `regex`: objects are passed through unchanged.
- `character_class`: objects are passed through unchanged.

- `character`: objects properly escaped for character classes.
- `list`: call `character_class_escape` on all elements of the list.
- `default`: coerce to `character` and `character_class_escape`.

`counts` *Counts*

Description

Functions to restrict a regex to a specific number

Usage

```
n_times(x, n, type = c("greedy", "lazy", "possessive"))

between(x, low, high, type = c("greedy", "lazy", "possessive"))

at_least(x, n, type = c("greedy", "lazy", "possessive"))

at_most(x, n, type = c("greedy", "lazy", "possessive"))
```

Arguments

`x`	A regex pattern.
`n`	An integer number
`type`	the type of match to perform. There are three match types 1. `greedy`: match the longest string. This is the default matching type. 2. `lazy`: match the shortest string. This matches the shortest string from the same anchor point, not necessarily the shortest global string. 3. `possessive`: match and don't allow backtracking
`low`	An integer number for the lower limit.
`high`	An integer number for the upper limit.

Functions

- `n_times`: `x` must occur exactly `n` times.
- `between`: `x` must occur between `low` and `high` times.
- `at_least`: `x` must occur at least `n` times.
- `at_most`: `x` must occur at most `n` times.

See Also

Other rex: `%or%`, `capture`, `character_class`, `group`, `lookarounds`, `not`, `rex`, `shortcuts`, `wildcards`

escape *Escape characters for a regex*

Description

Escape characters for a regex

Usage

```
escape(x)

## S3 method for class 'regex'
escape(x)

## S3 method for class 'character_class'
escape(x)

## S3 method for class 'character'
escape(x)

## Default S3 method:
escape(x)

## S3 method for class 'list'
escape(x)
```

Arguments

x	Object to escape.

Methods (by class)

- `regex`: Objects are simply passed through unchanged.
- `character_class`: Objects are surrounded by braces.
- `character`: Objects are properly escaped for regular expressions.
- `default`: default escape coerces to character and escapes.
- `list`: simply call escape on all elements of the list.

group *Create a grouped expression*

Description

This is similar to capture except that it does not store the value of the group. Best used when you want to combine several parts together and do not reference or extract the grouped value later.

Usage

```
group(...)
```

Arguments

...	shortcuts, R variables, text, or other **rex** functions.

See Also

capture for grouping with capturing. Perl 5 Extended Patterns http://perldoc.perl.org/perlre.html#Extended-Patterns

Other rex: %or%, capture, character_class, counts, lookarounds, not, rex, shortcuts, wildcards

lookarounds *Lookarounds*

Description

Lookarounds

Usage

```
x %if_next_is% y

x %if_next_isnt% y

x %if_prev_is% y

x %if_prev_isnt% y
```

Arguments

x	A regex pattern.
y	A regex pattern.

Details

These functions provide an interface to perl lookarounds.

Special binary functions are used to infer an ordering, since often you might wish to match a word / set of characters conditional on the start and end of that word.

- `%if_next_is%`: TRUE if x follows y
- `%if_next_isnt%`: TRUE if x does not follow y
- `%if_prev_is%`: TRUE if y comes before x
- `%if_prev_isnt%`: TRUE if y does not come before x

See Also

Perl 5 Documentation http://perldoc.perl.org/perlre.html#Extended-Patterns

Other rex: `%or%`, `capture`, `character_class`, `counts`, `group`, `not`, `rex`, `shortcuts`, `wildcards`

Examples

```
stopifnot(grepl(rex("crab" %if_next_is% "apple"), "crabapple", perl = TRUE))
stopifnot(grepl(rex("crab" %if_prev_is% "apple"), "applecrab", perl = TRUE))
stopifnot(grepl(rex(range("a", "e") %if_next_isnt% range("f", "g")),
  "ah", perl = TRUE))
stopifnot(grepl(rex(range("a", "e") %if_next_is% range("f", "i")),
  "ah", perl = TRUE))
```

---

not *Do not match*

Description

Do not match

Usage

```
not(..., type = c("greedy", "lazy", "possessive"))
```

Arguments

`...`	`shortcuts`, R variables, text, or other **rex** functions.
`type`	the type of match to perform. There are three match types 1. `greedy`: match the longest string. This is the default matching type. 2. `lazy`: match the shortest string. This matches the shortest string from the same anchor point, not necessarily the shortest global string. 3. `possessive`: match and don't allow backtracking

See Also

Other rex: %or%, capture, character_class, counts, group, lookarounds, rex, shortcuts, wildcards

---

`register_shortcuts` *Register the Rex shortcuts*

Description

If you are using rex in another package you need to call this function to register all of the rex shortcuts so that spurious NOTEs about global variables being generated during R CMD check.

Usage

```
register_shortcuts(pkg_name)
```

Arguments

pkg_name	the package to register the shortcuts in

`rex` *Generate a regular expression.*

Description

Generate a regular expression.

Usage

```
rex(..., env = parent.frame())
```

Arguments

...	shortcuts, R variables, text, or other **rex** functions.
env	environment to evaluate the rex expression in.

See Also

Other rex: %or%, capture, character_class, counts, group, lookarounds, not, shortcuts, wildcards

rex_mode *Toggles* **rex** *mode.*

Description

While within rex mode, functions used within the `rex` function are attached, so one can get e.g. auto-completion within editors.

Usage

```
rex_mode()
```

re_matches *Match function*

Description

Match function

Usage

```
re_matches(data, pattern, global = FALSE, options = NULL,
  locations = FALSE, ...)
```

Arguments

`data`	character vector to match against
`pattern`	regular expression to use for matching
`global`	use global matching
`options`	regular expression options
`locations`	rather than returning the values of the matched (or captured) string, return a `data.frame` of the match locations in the string.
`...`	options passed to regexpr or gregexpr

Value

if no captures, returns a logical vector the same length as the input character vector specifying if the relevant value matched or not. If there are captures in the regular expression, returns a `data.frame` with a column for each capture group. If `global` is `TRUE`, returns a list of `data.frames`.

See Also

`regexp` Section "Perl-like Regular Expressions" for a discussion of the supported options

Examples

```
string <- c("this is a", "test string")
re_matches(string, rex("test")) # FALSE FALSE

# named capture
re_matches(string, rex(capture(alphas, name = "first_word"), space,
  capture(alphas, name = "second_word")))
#   first_word second_word
# 1       this          is
# 2       test      string

# capture returns NA when it fails to match
re_matches(string, rex(capture("test")))
#      1
# 1 test
# 2 <NA>
```

`re_substitutes` *Substitute regular expressions in a string with another string.*

Description

Substitute regular expressions in a string with another string.

Usage

```
re_substitutes(data, pattern, replacement, global = FALSE, options = NULL,
  ...)
```

Arguments

`data`	character vector to substitute
`pattern`	regular expression to match
`replacement`	replacement text to use
`global`	substitute all occurrences
`options`	option flags
`...`	options passed to sub or gsub

See Also

`regexp` Section "Perl-like Regular Expressions" for a discussion of the supported options

Examples

```
string <- c("this is a Test", "string")
re_substitutes(string, "test", "not a test", options = "insensitive")
re_substitutes(string, "i", "x", global = TRUE)
re_substitutes(string, "(test)", "not a \\1", options = "insensitive")
```

shortcuts *Shortcuts*

Description

Commonly used character classes and regular expressions. These shortcuts are substituted inside `rex` calls.

Usage

```
shortcuts
```

Format

```
dot - \.
any - .
something - .+
anything - .*
start - ^
end - $
boundary - \b
non_boundary - \B
alnum - [:alnum:]
alpha - [:alpha:]
letter - [:alpha:]
blank - [:blank:]
cntrl - [:cntrl:]
digit - [:digit:]
number - [:digit:]
graph - [:graph:]
lower - [:lower:]
print - [:print:]
punct - [:punct:]
space - [:space:]
upper - [:upper:]
xdigit - [:xdigit:]
newline - \R
single_quote - '
double_quote - "
quote - '"
alnums - [[:alnum:]]+
alphas - [[:alpha:]]+
letters - [[:alpha:]]+
blanks - [[:blank:]]+
cntrls - [[:cntrl:]]+
digits - [[:digit:]]+
numbers - [[:digit:]]+
```

```
graphs - [[:graph:]]+
lowers - [[:lower:]]+
prints - [[:print:]]+
puncts - [[:punct:]]+
spaces - [[:space:]]+
uppers - [[:upper:]]+
xdigits - [[:xdigit:]]+
newlines - \R+
single_quotes - [']+
double_quotes - [”]+
quotes - ['”]+
any_alnums - [[:alnum:]]*
any_alphas - [[:alpha:]]*
any_letters - [[:alpha:]]*
any_blanks - [[:blank:]]*
any_cntrls - [[:cntrl:]]*
any_digits - [[:digit:]]*
any_numbers - [[:digit:]]*
any_graphs - [[:graph:]]*
any_lowers - [[:lower:]]*
any_prints - [[:print:]]*
any_puncts - [[:punct:]]*
any_spaces - [[:space:]]*
any_uppers - [[:upper:]]*
any_xdigits - [[:xdigit:]]*
any_newlines - \R*
any_single_quotes - [']*
any_double_quotes - [”]*
any_quotes - ['”]*
non_alnum - ^[:alnum:]
non_alpha - ^[:alpha:]
non_letter - ^[:alpha:]
non_blank - ^[:blank:]
non_cntrl - ^[:cntrl:]
non_digit - ^[:digit:]
non_number - ^[:digit:]
non_graph - ^[:graph:]
non_lower - ^[:lower:]
non_print - ^[:print:]
non_punct - ^[:punct:]
non_space - ^[:space:]
non_upper - ^[:upper:]
non_xdigit - ^[:xdigit:]
non_newline - ^\R
non_single_quote - ^'
non_double_quote - ^”
non_quote - ^'”
non_alnums - [^[:alnum:]]+
```

```
non_alphas - [^[:alpha:]]+
non_letters - [^[:alpha:]]+
non_blanks - [^[:blank:]]+
non_cntrls - [^[:cntrl:]]+
non_digits - [^[:digit:]]+
non_numbers - [^[:digit:]]+
non_graphs - [^[:graph:]]+
non_lowers - [^[:lower:]]+
non_prints - [^[:print:]]+
non_puncts - [^[:punct:]]+
non_spaces - [^[:space:]]+
non_uppers - [^[:upper:]]+
non_xdigits - [^[:xdigit:]]+
non_newlines - ^\R+
non_single_quotes - [^']+
non_double_quotes - [^"]+
non_quotes - [^'"]+
any_non_alnums - [^[:alnum:]]*
any_non_alphas - [^[:alpha:]]*
any_non_letters - [^[:alpha:]]*
any_non_blanks - [^[:blank:]]*
any_non_cntrls - [^[:cntrl:]]*
any_non_digits - [^[:digit:]]*
any_non_numbers - [^[:digit:]]*
any_non_graphs - [^[:graph:]]*
any_non_lowers - [^[:lower:]]*
any_non_prints - [^[:print:]]*
any_non_puncts - [^[:punct:]]*
any_non_spaces - [^[:space:]]*
any_non_uppers - [^[:upper:]]*
any_non_xdigits - [^[:xdigit:]]*
any_non_newlines - ^\R*
any_non_single_quotes - [^']*
any_non_double_quotes - [^"]*
any_non_quotes - [^'"]*
```

Details

`names(shortcuts)` will give you the full list of available shortcuts.

See Also

Other rex: `%or%`, `capture`, `character_class`, `counts`, `group`, `lookarounds`, `not`, `rex`, `wildcards`

`single_shortcuts` *Single shortcuts*

Description

Each of these shortcuts has both a plural (-s) and inverse (non_) form.

Usage

```
single_shortcuts
```

Format

```
alnum - [:alnum:]
alpha - [:alpha:]
letter - [:alpha:]
blank - [:blank:]
cntrl - [:cntrl:]
digit - [:digit:]
number - [:digit:]
graph - [:graph:]
lower - [:lower:]
print - [:print:]
punct - [:punct:]
space - [:space:]
upper - [:upper:]
xdigit - [:xdigit:]
newline - \R
single_quote - '
double_quote - "
quote - '"
```

`wildcards` *Wildcards*

Description

Wildcards

Usage

```
zero_or_more(..., type = c("greedy", "lazy", "possessive"))

one_or_more(..., type = c("greedy", "lazy", "possessive"))

maybe(..., type = c("greedy", "lazy", "possessive"))
```

Arguments

...	shortcuts, R variables, text, or other **rex** functions.
type	the type of match to perform. There are three match types 1. greedy: match the longest string. This is the default matching type. 2. lazy: match the shortest string. This matches the shortest string from the same anchor point, not necessarily the shortest global string. 3. possessive: match and don't allow backtracking

Functions

- zero_or_more: match ... zero or more times.
- one_or_more: match ... one or more times.
- maybe: match ... zero or one times.

See Also

Other rex: %or%, capture, character_class, counts, group, lookarounds, not, rex, shortcuts

---

%or% *Or*

Description

The special binary function %or% can be used to specify a set of optional matches.

describeIn or regular function can also be used, useful for more than 2 arguments.

Usage

```
x %or% y

or(...)
```

Arguments

x	A string.
y	A string.
...	shortcuts, R variables, text, or other **rex** functions.

See Also

Other rex: capture, character_class, counts, group, lookarounds, not, rex, shortcuts, wildcards

Index