THE STRINGR PACKAGE

by Gwen Rino

CSX 415.1

08 April 2018

STRINGR

has four families of functions

- CHARACTER MANIPULATION

STRINGR

has four families of functions

- CHARACTER MANIPULATION
- WHITESPACE TOOLS

STRINGR

has four families of functions

- CHARACTER MANIPULATION
- WHITESPACE TOOLS
- LOCALE SENSITIVE OPERATIONS

STRINGR

has four families of functions

- CHARACTER MANIPULATION
- WHITESPACE TOOLS
- LOCALE SENSITIVE OPERATIONS
- PATTERN MATCHING FUNCTIONS

CHARACTER MANIPULATION

str_length()

str_length() returns the length of a string.

```
str_length("abcde")
```

```
## [1] 5
```

str_sub()

str_sub() accesses individual characters in a string.

```
vec <- c("a string", "another string")

# Returns the third character in each string
str_sub(vec, 3, 3)
```

```
## [1] "s" "o"
```

```
# Returns characters 3-5 in each string
str_sub(vec, 3, 5)
```

```
## [1] "str" "oth"
```

```
# Count from right using negative numbers
str_sub(vec, -4, -1)
```

```
## [1] "ring" "ring"
```

str_sub()

You can use `str_sub()` to modify strings.

```
vec <- c("a string", "another string")

str_sub(vec, 3, 3) <- "X"
vec
```

```
## [1] "a Xtring"       "anXther string"
```

str_dup()

You can use `str_dup()` to duplicate strings.

```
print(vec)
```

```
## [1] "a Xtring"        "anXther string"
```

```
str_dup(vec, c(2,3))
```

```
## [1] "a Xtringa Xtring"
## [2] "anXther stringanXther stringanXther string"
```

WHITESPACE TOOLS

str_pad()

str_pad() pads a string to a given length by adding white space.

```
x <- c("z", "abcedfg")
str_pad(x, 10) # Default padding is on the left
```

```
## [1] "         z" "   abcedfg"
```

```
str_pad(x, 10, "right")
```

```
## [1] "z         " "abcedfg   "
```

```
str_pad(x, 10, "both")
```

```
## [1] "    z     " " abcedfg  "
```

str_pad()

You can pad with characters other than spaces using the pad argument.

```
x <- c("z", "abcedfg")

str_pad(x, 3, pad="X")
```

```
## [1] "XXz"     "abcedfg"
```

*Notice that padding to a length $<$ the length of the string does not truncate the string!

str_trunc()

You can truncate a string to a given length (including a 3 character ellipsis) using `str_trunc()`.

```
str_trunc("Thisstringisquitelong", 13, "right")
```

```
## [1] "Thisstring..."
```

str_trim()

The opposite of `str_pad()` is `str_trim()`. It trims leading and trailing white space.

```
y <- c("   a", "b   ", "  c  ")
str_trim(y) # Default trims white space from both sides
```

```
## [1] "a" "b" "c"
```

```
str_trim(y, "left")
```

```
## [1] "a"    "b   " "c  "
```

```
str_trim(y, "right")
```

```
## [1] "   a" "b"     "  c"
```

str_wrap()

You can use `str_wrap()` to wrap a paragraph of text, finding whitespace breaks such that the width of each line is as similar to the given argument as possible.

```
jabberwocky <- "`Twas brillig, and the slithy toves did gyr
in the wabe: All mimsy were the borogoves and the mome rath

str_wrap(jabberwocky, width = 40)
```

```
## [1] "`Twas brillig, and the slithy toves did\ngyre and g
```

```
cat(str_wrap(jabberwocky, width = 40))
```

```
## `Twas brillig, and the slithy toves did
## gyre and gimble in the wabe: All mimsy
## were the borogoves and the mome raths
## outgrabe.
```

LOCALE SENSITIVE OPERATIONS

A few stringr functions are locale-sensitive, which means that they can perform differently to accommodate different languages.

The default is always English. You can accommodate different languages by setting the `locale` argument to a two letter ISO-639-1 code.

You can see a complete list of available locales by running `stringi::stri_locale_list()`.

str_sort() and str_order()

For example, in Lithuanian, y comes between i and j.

```
str_order(letters)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [24] 24 25 26
```

```
str_order(letters, locale = "lt")
```

```
##  [1]  1  2  3  4  5  6  7  8  9 25 10 11 12 13 14 15 16
## [24] 23 24 26
```

```
str_sort(letters, locale = "lt")
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "y" "j" "k" "l"
## [18] "q" "r" "s" "t" "u" "v" "w" "x" "z"
```

str_to_lower()

Another example: Turkish has two different letters 'i', one with and one without a dot.

```
x <- "I like horses."
str_to_lower(x)
```

```
## [1] "i like horses."
```

```
str_to_lower(x, "tr")
```

```
## [1] "ı like horses."
```

PATTERN MATCHING

stringr includes a number of functions that are used to process a character vector of strings for matches with a single pattern.

```r
# Character vector to process
strings <- c(
  "apple",
  "219 733 8965",
  "329-293-8753",
  "Work: 579-499-7527; Home: 543.355.3679"
)

# Pattern to match
# (This is a regular expression designed to match US phone
phone <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
```

str_detect()

`str_detect()` detects the presence or absence of a pattern and returns a logical vector.

```
# Does each string contain a phone number?
str_detect(strings, phone)
```

```
## [1] FALSE  TRUE  TRUE  TRUE
```

str_subset()

`str_subset()` returns the elements of a character vector that match a pattern.

```
# Which strings contain phone numbers?
str_subset(strings, phone)
```

```
## [1] "219 733 8965"
## [2] "329-293-8753"
## [3] "Work: 579-499-7527; Home: 543.355.3679"
```

str_count()

`str_count()` counts the number of matches in each string.

```
# How many phone numbers are in each string?
str_count(strings, phone)
```

```
## [1] 0 1 1 2
```

`str_locate()` locates the first position of a pattern and returns a matrix with start and end positions as columns.

```
# Where in each string is the first phone number located?
str_locate(strings, phone)
```

```
##       start end
## [1,]     NA  NA
## [2,]      1  12
## [3,]      1  12
## [4,]      7  18
```

`str_locate_all()` locates all matches and returns a list of matrices.

```
# Where are all the phone numbers located?
str_locate_all(strings, phone)
```

```
## [[1]]
##      start end
##
## [[2]]
##      start end
## [1,]     1  12
##
## [[3]]
##      start end
## [1,]     1  12
##
## [[4]]
##      start end
## [1,]     7  18
```

`str_extract()` extracts text corresponding to the first match, returning a character vector.

```
# What is the first phone number in each string?
str_extract(strings, phone)
```

```
## [1] NA               "219 733 8965" "329-293-8753" "579-49
```

You can also do `str_extract_all()`, which returns a list of character vectors.

`str_match()` extracts capture groups from the first match formed by () in the regular expression. It returns a character matrix with one column for the complete match and one column for each group.

```
# Pull out the three components of the first match in each
str_match(strings, phone)
```

```
##      [,1]              [,2]  [,3]  [,4]
## [1,] NA                NA    NA    NA
## [2,] "219 733 8965"    "219" "733" "8965"
## [3,] "329-293-8753"    "329" "293" "8753"
## [4,] "579-499-7527"    "579" "499" "7527"
```

You can also do `str_match_all()`, which extracts capture groups from all matches and returns a list of character matrices.

`str_replace()` replaces the first matched pattern and returns a character vector.

```
str_replace(strings, phone, "XXX-XXX-XXXX")
```

```
## [1] "apple"
## [2] "XXX-XXX-XXXX"
## [3] "XXX-XXX-XXXX"
## [4] "Work: XXX-XXX-XXXX; Home: 543.355.3679"
```

`str_replace_all()` replaces all matches.

`str_split()` splits a string into a variable number of pieces based on a pattern and returns a list of character vectors.

```
str_split("a-b-c", "-")
```

```
## [[1]]
## [1] "a" "b" "c"
```

`str_split_fixed()` splits a string into a fixed number of pieces and returns a character matrix.

```
str_split_fixed("a-b-c", "-", n = 2)
```

```
##      [,1] [,2]
## [1,] "a"  "b-c"
```