# THE STRINGR PACKAGE

by Gwen Rino

CSX 415.1

08 April 2018

The **stringr** package provides a set of internally consistent tools for working with character strings in R.

**stringr** is good for

- detecting matches in strings
- subsetting strings
- managing lengths of strings
- mutating strings
- joining and splitting strings
- ordering strings

**stringr** is good for

- **detecting matches in strings**
- subsetting strings
- managing lengths of strings
- mutating strings
- joining and splitting strings
- ordering strings

# DETECTING MATCHES

**stringr** offers four functions for **detecting matches**, all with the same two arguments (string, pattern).

- `str_detect()`
- `str_which()`
- `str_count()`
- `str_locate()`

# DETECTING MATCHES

**str_detect()** detects the presence of a pattern match in a string.

```
str_detect(fruit, "a")
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TR
##  [12]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FAL
##  [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FAL
##  [34]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FAL
##  [45] FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FAL
##  [56]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TR
##  [67] FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TR
##  [78]  TRUE FALSE  TRUE
```

# DETECTING MATCHES

**str_which()** finds the indexes of strings that contain a pattern match.

```
str_which(fruit, "a")
```

```
##  [1]  1  2  3  4  7  8  9 12 13 14 15 21 23 24 25 26 27
## [24] 40 43 46 48 49 51 54 55 56 57 58 59 61 62 64 66 68
## [47] 76 77 78 80
```

# DETECTING MATCHES

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

```
str_count(fruit, "a")
```

```
##  [1] 1 1 2 3 0 0 1 2 1 0 0 1 2 2 1 0 0 0 0 0 1 0 1 1 1 1
## [36] 2 0 0 1 1 0 0 1 0 0 1 0 2 1 0 1 0 0 1 1 3 1 1 1 0 1
## [71] 1 0 2 2 1 1 2 1 0 1
```

# DETECTING MATCHES

`str_locate()` locates the positions of the first pattern match in a string. (Also `str_locate_all()`)

```
str_locate(fruit, "a")
```

```
##        start end
##   [1,]     1   1
##   [2,]     1   1
##   [3,]     1   1
##   [4,]     2   2
##   [5,]    NA  NA
##   [6,]    NA  NA
##   [7,]     3   3
##   [8,]     3   3
##   [9,]     9   9
##  [10,]    NA  NA
##  [11,]    NA  NA
##  [12,]     4   4
```

**stringr** is good for

- detecting matches in strings
- **subsetting strings**
- managing lengths of strings
- mutating strings
- joining and splitting strings
- ordering strings

# SUBSETTING STRINGS

**stringr** offers four functions for **subsetting strings**.

- `str_sub()`
- `str_subset()`
- `str_extract()`
- `str_match()`

# SUBSETTING STRINGS

**str_sub()** extracts substrings from a character vector. Arguments are (string, start, end).

```
str_sub(fruit, 1, 3)
```

```
##  [1] "app" "apr" "avo" "ban" "bel" "bil" "bla" "bla" "bl
## [12] "bre" "can" "can" "che" "che" "chi" "cle" "clo" "co
## [23] "cur" "dam" "dat" "dra" "dur" "egg" "eld" "fei" "fi
## [34] "gra" "gra" "gua" "hon" "huc" "jac" "jam" "juj" "ki
## [45] "lim" "loq" "lyc" "man" "man" "mul" "nec" "nut" "ol
## [56] "pap" "pas" "pea" "pea" "per" "phy" "pin" "plu" "po
## [67] "qui" "rai" "ram" "ras" "red" "roc" "sal" "sat" "st
## [78] "tan" "ugl" "wat"
```

# SUBSETTING STRINGS

**`str_subset()`** returns only the strings that contain a pattern match. Arguments are (string, pattern).

```
str_subset(fruit, "b")
```

```
##  [1] "banana"        "bell pepper"  "bilberry"     "black
##  [5] "blackcurrant" "blood orange" "blueberry"    "boyse
##  [9] "breadfruit"   "cloudberry"   "cranberry"    "cucur
## [13] "elderberry"   "goji berry"   "gooseberry"   "huckl
## [17] "jambul"       "jujube"       "mulberry"     "rambu
## [21] "raspberry"    "salal berry"  "strawberry"
```

# SUBSETTING STRINGS

**`str_extract()`** returns the first pattern match found in each string, as a vector. Arguments are (string, pattern). (Also **`str_extract_all()`**)

```
str_extract(fruit, "be")
```

```
##   [1] NA   NA   NA   NA   "be" "be" "be" NA   NA   "be" "
##  [15] NA   NA   NA   NA   "be" NA   "be" "be" NA   NA   N
##  [29] "be" NA   NA   "be" "be" NA   NA   NA   NA   "be" N
##  [43] NA   NA   NA   NA   NA   NA   NA   "be" NA   NA   N
##  [57] NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   N
##  [71] NA   NA   "be" NA   NA   "be" NA   NA   NA   NA
```

# SUBSETTING STRINGS

**str_match()** returns the first pattern match found in each string, as a matrix with a column for each () group in pattern. Arguments are (string, pattern). (Also **str_match_all()**)

```
str_match(sentences, "(a|the) ([^ ]+)") # regex for word "
```

```
##          [,1]           [,2]  [,3]
##    [1,] "the smooth"   "the" "smooth"
##    [2,] "the sheet"    "the" "sheet"
##    [3,] "the depth"    "the" "depth"
##    [4,] "a chicken"    "a"   "chicken"
##    [5,] NA             NA    NA
##    [6,] NA             NA    NA
##    [7,] "the parked"   "the" "parked"
##    [8,] NA             NA    NA
##    [9,] NA             NA    NA
##   [10,] NA             NA    NA
##   [11,] "the sun"      "the" "sun"
```

**stringr** is good for

- detecting matches in strings
- subsetting strings
- **managing lengths of strings**
- mutating strings
- joining and splitting strings
- ordering strings

# MANAGING LENGTHS

**stringr** offers four functions for **managing lengths** of strings.

- `str_length()`
- `str_pad()`
- `str_trunc()`
- `str_trim()`

# MANAGING LENGTHS

**`str_length()`** returns the width of strings (i.e. the number of characters).

```
str_length(fruit)
```

```
##  [1]  5  7  7  6 11  8 10 12 12  9 11 10 12 10  9  6 12
## [24]  6  4 11  6  8 10  6  3 10 10  5 10  5  8 11  9  6
## [47]  6  9  5  8  9  3  5  6  6  6 12  5  4  9  8  9  4
## [70]  9 10 10 11  7 10 10  9  9 10 10
```

# MANAGING LENGTHS

**str_pad** pads strings to a constant width. Arguments are (string, width, side = c("left", "right", "both"), pad = " ").

```
str_pad(fruit, 15, "left")
```

```
##  [1] "          apple"   "        apricot"   "          av
##  [4] "         banana"   "    bell pepper"   "         bi
##  [7] "     blackberry"   "   blackcurrant"   "   blood o
## [10] "      blueberry"   "    boysenberry"   "      brea
## [13] "   canary melon"   "     cantaloupe"   "       che
## [16] "         cherry"   "   chili pepper"   "      cleme
## [19] "     cloudberry"   "        coconut"   "       cra
## [22] "       cucumber"   "        currant"   "         c
## [25] "           date"   "    dragonfruit"   "         c
## [28] "       eggplant"   "     elderberry"   "         f
## [31] "            fig"   "     goji berry"   "      goose
## [34] "          grape"   "     grapefruit"   "
## [37] "       honeydew"   "    huckleberry"   "       jack
```

# MANAGING LENGTHS

**str_trunc** truncates the width of strings, replacing content with ellipsis. Arguments are (string, width, side = c("left", "right", "both"), ellipsis = "...").

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

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

# MANAGING LENGTHS

**str_trim** trims white space from the start or end of a string. Arguments are (string, side = c("left", "right", "both")).

```
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"
```

# 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.

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

# Pattern to match (a regex to match US phone numbers)
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"
```