Week 7: Strings

EMSE 6574, Section 11

John Helveston
October 07, 2019

Quiz 3

20 minutes

- No calculators
- No notes
- No books
- No computers
- No phones

Announcements

1) You have **3** weeks for HW 4

2) Exam 1:

- 100 minutes (1 hr 40 min)
- Weeks 1-6 (no strings)
- No Turtle Graphics

3) **This week**: Instead of office hours, we'll have an exam review

Install the `stringr` library

```r
install.packages("stringr")
library(stringr)
```

Making a string

Single or double quotes - they both work:

```r
cat("This is a string")
```

```
## This is a string
```

```r
cat('This is a string')
```

```
## This is a string
```

Use them where it makes sense, e.g.:

```r
cat("It's a boy!")
```

```
## It's a boy!
```

```r
cat('I said, "Hi!"')
```

```
## I said, "Hi!"
```

Making a string

What if a string has both `'` and `"` symbols?

Example: `It's nice to say, "Hi!"`

```
cat("It's nice to say, "Hi!"")
```

```
## Error: <text>:1:25: unexpected symbol
## 1: cat("It's nice to say, "Hi
##                              ^
```

```
cat('It's nice to say, "Hi!"'')
```

```
## Error: <text>:1:9: unexpected symbol
## 1: cat('It's
##              ^
```

"Escaping" to the rescue!

Use the `\` symbol to "escape" a literal symbol:

```
cat("It's nice to say, \"Hi!\"") # Double quote
```

```
## It's nice to say, "Hi!"
```

```
cat('It\'s nice to say, "Hi!"') # Single quote
```

```
## It's nice to say, "Hi!"
```

```
cat('This\nthat') # New line
```

```
## This
## that
```

```
cat('This\tthat') # Tab space
```

```
## This    that
```

```
cat('This\\that') # Backslash
```

```
## This\that
```

String constants

R has a few built-in string constants:

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
letters
```

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

```
month.name
```

```
##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"
```

```
month.abb
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
## [12] "Dec"
```

String constants

The **stringr** library has some longer string constants:

```
head(words)
```

```
## [1] "a"        "able"     "about"    "absolute" "accept"   "account"
```

```
length(words)
```

```
## [1] 980
```

```
head(sentences)
```

```
## [1] "The birch canoe slid on the smooth planks."
## [2] "Glue the sheet to the dark blue background."
## [3] "It's easy to tell the depth of a well."
## [4] "These days a chicken leg is a rare dish."
## [5] "Rice is often served in round bowls."
## [6] "The juice of lemons makes fine punch."
```

```
head(fruit)
```

```
## [1] "apple"       "apricot"     "avocado"     "banana"      "bell pepper"
## [6] "bilberry"
```

Main **stringr** functions

Function	**Description**
`str_to_lower()`	converts string to lower case
`str_to_upper()`	converts string to upper case
`str_to_title()`	converts string to title case
`str_length()`	number of characters
`str_sub()`	extracts substrings
`str_dup()`	duplicates characters
`str_trim()`	removes leading and trailing whitespace
`str_pad()`	pads a string
`str_c()`	string concatenation
`str_split()`	split a string into a vector
`str_sort()`	sort a string alphabetically
`str_order()`	get the order of a sorted string
`str_detect()`	match a string in another string
`str_replace()`	replace a string in another string

Case conversion

We saw these in HW 1!

```
x <- "Want to hear a joke about paper? Never mind, it's tearable."
```

```
str_to_lower(x)
```

```
## [1] "want to hear a joke about paper? never mind, it's tearable."
```

```
str_to_upper(x)
```

```
## [1] "WANT TO HEAR A JOKE ABOUT PAPER? NEVER MIND, IT'S TEARABLE."
```

```
str_to_title(x)
```

```
## [1] "Want To Hear A Joke About Paper? Never Mind, It's Tearable."
```

```
library(tools)
toTitleCase(x)
```

```
## [1] "Want to Hear a Joke About Paper? Never Mind, It's Tearable."
```

Comparing strings

```r
a <- "Apples"
b <- "apples"
a == b
```

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

Use `str_to_lower()` or `str_to_upper()` to **ignore case**:

```r
str_to_lower(a) == str_to_lower(b)
```

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

```r
str_to_upper(a) == str_to_upper(b)
```

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

Get the number of characters in a string

What will this return?

```
length("hello world")
```

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

To get the # of characters, use `str_length()`:

```
str_length("hello world")
```

```
## [1] 11
```

```
str_length("The quick brown fox jumped over the lazy dog")
```

```
## [1] 44
```

```
str_length(" ")
```

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

```
str_length("")
```

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

Access characters by their index

`str_sub()`

```
str_sub("Apple", 1, 3)
```

```
## [1] "App"
```

```
str_sub("Apple", -3, -1) # Negative numbers count backwards from the end
```

```
## [1] "ple"
```

```
str_sub("hi", 1, 5) # If string is too short, R won't error
```

```
## [1] "hi"
```

Use `str_sub()` to modify a string:

```
x <- 'abcdef'
str_sub(x, 1, 3) <- 'ABC'
```

```
x
```

```
## [1] "ABCdef"
```

Repeat a string

`str_dup()`

```r
str_dup("hola", 3)
```

```
## [1] "holaholahola"
```

Note the difference with `rep()`:

```r
rep("hola", 3)
```

```
## [1] "hola" "hola" "hola"
```

stringr functions work on vectors

```
x <- c("apples", "oranges")
x
```

```
## [1] "apples"  "oranges"
```

Get the first 3 letters in each string:

```
str_sub(x, 1, 3)
```

```
## [1] "app" "ora"
```

Duplicate each string twice

```
str_dup(x, 2)
```

```
## [1] "applesapples"   "orangesoranges"
```

```
str_to_upper(x)
```

```
## [1] "APPLES"  "ORANGES"
```

Quick practice: Think-Pair-Share

1) Create this string object: `x <- 'thisIsGoodPractice'`

2) Use `stringr` functions to transform `x` into the following strings:

- `'thisIsGood'`
- `'practice'`
- `'GOOD'`
- `'thisIsGoodPracticethisIsGoodPractice'`
- `'thisthisthis'`
- `'GOODGOODGOOD'`

Hint: You'll need these:

- `str_to_lower()`
- `str_to_upper()`
- `str_sub()`
- `str_dup()`

Hint: You may want to create intermediate variables

Remove excess white space

`str_trim()`

```
x <- "         aStringWithSpace        "
x
```

```
## [1] "         aStringWithSpace        "
```

```
str_trim(x) # Trims both sides by default
```

```
## [1] "aStringWithSpace"
```

```
str_trim(x, side = "left") # Only trim left side
```

```
## [1] "aStringWithSpace        "
```

```
str_trim(x, side = "right") # Only trim right side
```

```
## [1] "         aStringWithSpace"
```

Add white space (or other characters)

`str_pad()`

```
x <- "hello"
x
```

```
## [1] "hello"
```

```
str_pad(x, width = 10) # Inserts pad on left by default
```

```
## [1] "     hello"
```

```
str_pad(x, width = 10, side = "both") # Pad both sides
```

```
## [1] "  hello   "
```

Pad with a different character:

```
str_pad(x,  width = 10, side = "both", pad = '*')
```

```
## [1] "**hello***"
```

Combine strings into one string

`str_c()`

```
str_c('x', 'y', 'z')
```

```
## [1] "xyz"
```

Control separation with `sep` argument:

```
str_c('x', 'y', 'z', sep = "-")
```

```
## [1] "x-y-z"
```

Note the difference with *vectors* of strings:

```
x <- c('x', 'y', 'z')
str_c(x)
```

```
## [1] "x" "y" "z"
```

To make a single string from a vector of strings, use `collapse`:

```
str_c(x, collapse = "")
```

```
## [1] "xyz"
```

`str_c` works with function logic

```r
printGreeting <- function(name, timeOfDay, isBirthday) {
    greeting <- str_c(
        "Good ", timeOfDay, " ", name,
            if (isBirthday) {
                ", and HAPPY BIRTHDAY!"
            } else {
                '.'
            }
        )
    cat(greeting)
}
```

What do you think this will print?

```r
printGreeting('John', 'morning', isBirthday = FALSE)
printGreeting('John', 'morning', isBirthday = TRUE)
```

```
## Good morning John.
```

```
## Good morning John, and HAPPY BIRTHDAY!
```

Split a string into multiple strings

`str_split()`

```
x <- 'This string has spaces-and-dashes'
x
```

```
## [1] "This string has spaces-and-dashes"
```

```
str_split(x, " ") # Split on the spaces
```

```
## [[1]]
## [1] "This"              "string"            "has"              
## [4] "spaces-and-dashes"
```

```
str_split(x, "-") # Split on the dashes
```

```
## [[1]]
## [1] "This string has spaces" "and"                   
## [3] "dashes"
```

What's with the `[[1]]` thing?

`str_split()` returns a `list` of vectors

```
x <- c('babble', 'scrabblebabble')
str_split(x, 'bb')
```

```
## [[1]]
## [1] "ba" "le"
##
## [[2]]
## [1] "scra" "leba" "le"
```

If you're only splitting one string, add `[[1]]` to get the first vector:

```
str_split('hooray', 'oo')[[1]]
```

```
## [1] "h"   "ray"
```

Common splits (memorize these)

Splitting on `""` breaks a string into *characters*:

```r
str_split("apples", "")[[1]]
```

```
## [1] "a" "p" "p" "l" "e" "s"
```

Splitting on `" "` breaks a *sentence* into words:

```r
x <- "If you want to view paradise, simply look around and view it"
str_split(x, " ")[[1]]
```

```
##  [1] "If"        "you"       "want"      "to"        "view"
##  [6] "paradise," "simply"    "look"      "around"    "and"
## [11] "view"      "it"
```

Quick practice: Think-Pair-Share

1) Create the following objects:

```
x <- 'this_is_good_practice'
y <- c('hello', 'world')
```

2) Use `stringr` functions to transform `x` and `y` into the following:

- `"hello world"`
- `"***hello world***"`
- `c("this", "is", "good", "practice")`
- `"this is good practice"`
- `"hello world, this is good practice"`

Hint: Create intermediate objects! And use these:

- `str_trim()`
- `str_pad()`
- `str_c()`
- `str_split()`

Practice: Think-Pair-Share

`getUniqueChars(s)`

Write a function that takes a single string, `s`, and returns an *alphabetically sorted* vector of the unique characters the string below. All letters should be lowercase (so `"A"` should be treated the same as `"a"`).

Example:

```
s <- 'babbleScrabbleApple'
getUniqueChars(s) == c("a","b","c","e","l","p","r","s")
```

Hints:

- Use `str_split()` to break a string into characters
- Check out the `unique()` function

```
getUniqueChars <- function(s) {
    s <- str_to_lower(s)
    chars <- str_split(s, "")[[1]] # Split the string into characters
    return(str_sort(unique(chars)))
}
```

5 minute break - stand up, move around,

5 minutes

Sort string vectors alphabetically

`str_sort()`

```
x <- c('Y', 'M', 'C', 'A')
x
```

```
## [1] "Y" "M" "C" "A"
```

```
str_sort(x)
```

```
## [1] "A" "C" "M" "Y"
```

```
str_sort(x, decreasing = TRUE)
```

```
## [1] "Y" "M" "C" "A"
```

```
str_order(x)
```

```
## [1] 4 3 2 1
```

```
x[str_order(x)]
```

```
## [1] "A" "C" "M" "Y"
```

Detect if pattern is in string

`str_detect(string, pattern)`

```r
tenFruit <- fruit[1:10]
tenFruit
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"      
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"
```

```r
str_detect(tenFruit, "berry")
```

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

Count how many have the string `"berry"`:

```r
sum(str_detect(tenFruit, "berry"))
```

```
## [1] 3
```

Count number of times pattern appears

`str_count(string, pattern)`

```r
x <- c("apple", "banana", "pear")
str_count(x, "a")
```

```
## [1] 1 3 1
```

Note the difference with `str_detect()`:

```r
str_detect(x, "a")
```

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

Detect if string *starts* with pattern

Example: Which fruit *start* with "a"?

```
tenFruit <- fruit[1:10]
tenFruit
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"
```

Wrong:

```
str_detect(tenFruit, "a")
```

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

Right:

```
str_detect(tenFruit, "^a")
```

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

Detect if string *ends* with pattern

Example: Which fruit *end* with an "e"?

```
tenFruit <- fruit[1:10]
tenFruit
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"      
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"
```

Wrong:

```
str_detect(tenFruit, "e")
```

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

Right:

```
str_detect(tenFruit, "e$")
```

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

Trick to remember this

If you *start* with power (`^`), you'll *end* up with money (`$`).

```
tenFruit
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"
```

```
str_detect(tenFruit, "^a")
```

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

```
str_detect(tenFruit, "e$")
```

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

Quick practice: Think-Pair-Share

1) For these questions, we'll use the `fruit` vector:

```
head(fruit)
```

```
## [1] "apple"       "apricot"     "avocado"     "banana"      "bell pepper"
## [6] "bilberry"
```

2) Use `stringr` functions to answer the following questions about the `fruit` vector:

- How many fruit have the string `"rr"` in it?
- Which fruit end with string `"fruit"`?
- Which fruit contain more than one `"o"` character?

Hint: You'll need these:

- `str_detect()`
- `str_count()`

Replace matched strings with new string

`str_replace(string, pattern, replacement)`

```r
x <- c("apple", "pear", "banana")
```

```r
str_replace(x, "a", "-")
```

```
## [1] "-pple"  "pe-r"   "b-nana"
```

```r
str_replace_all(x, "a", "-")
```

```
## [1] "-pple"  "pe-r"   "b-n-n-"
```

Practice Redux

Remember this task earlier?

```
x <- 'this_is_good_practice'
```

Convert `x` into: `"this is good practice"`

We did this earlier:

```
str_c(str_split(x, "_")[[1]], collapse = " ")
```

```
## [1] "this is good practice"
```

But now we can do this!

```
str_replace_all(x, "_", " ")
```

```
## [1] "this is good practice"
```

Group practice

20 minutes - In groups of 4, write the following functions:

1) `reverseString(s)`

Write a function that returns the string in reverse order. So if `s` equals `"abcde"`, `reverseString(s)` should equal `"edcba"`. You may assume that `s` only contains upper and/or lower case letters, but your solution must correctly return capital letters in their appropriate order. Here's some test cases:

- `reverseString("aWordWithCaps") == "spaChtiWdroWa"`
- `reverseString("abcde") == "edcba"`
- `reverseString("") == ""`

2) `isPalindrome(s)`

Write a function that returns `TRUE` if the string `s` is a Palindrome and `FALSE` otherwise. The string `s` can contains any letter, number, or symbol, but it will be a character data type. Here's some test cases:

- `isPalindrome("abcba") == TRUE`
- `isPalindrome("abcb") == FALSE`
- `isPalindrome("321123") == TRUE`

Group practice

20 minutes - In groups of 4, write the following functions:

1) `sortString(s)`

Write the function `sortString(s)` that takes a string `s` and returns back an alphabetically sorted string. Assume that `s` only contains upper and/or lower case letters. Here's some test cases:

- `sortString("cba") == "abc"`
- `sortString("abedhg") == "abdegh"`
- `sortString("AbacBc") == "aAbBcc"`

2) `areAnagrams(s1, s2)`

Write the function `areAnagrams(s1, s2)` that takes two strings, `s1` and `s2`, and returns `TRUE` if the strings are anagrams, and `FALSE` otherwise. Treat lower and upper case as the same letters. Here's some test cases:

- `areAnagrams("", "") == TRUE`
- `areAnagrams("TomMarvoloRiddle", "IAmLordVoldemort") == TRUE`
- `areAnagrams("aabbccdd", "bbccddee") == FALSE`