Week 7: *Strings*

EMSE 4574: Intro to Programming for Analytics

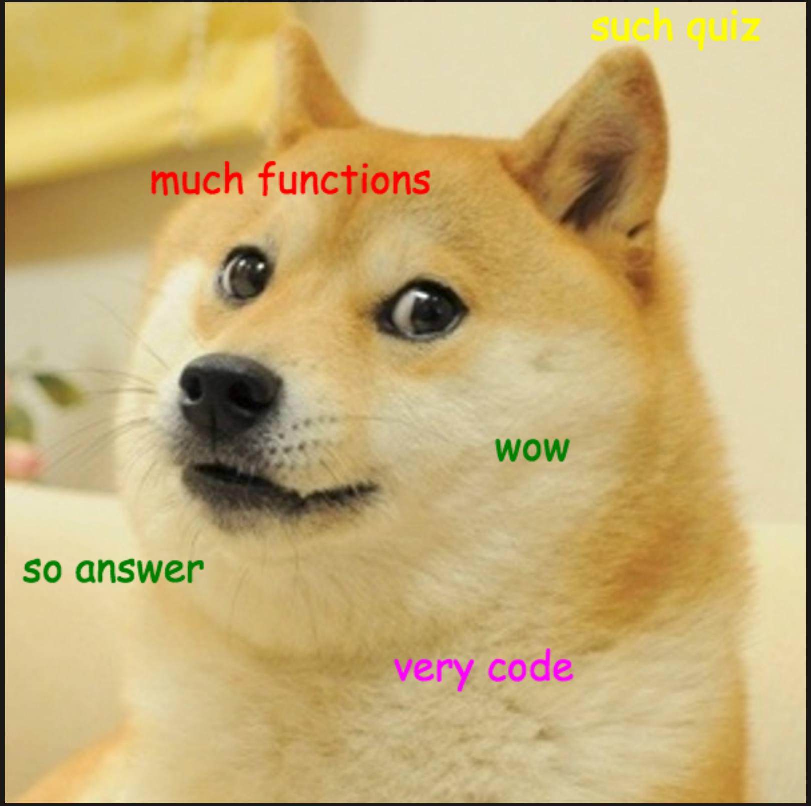
John Paul Helveston

October 13, 2020

Quiz 4

- Go to `#classroom` channel in Slack for link

05:00



Week 7: *Strings*

1. Making strings
2. Case conversion & substrings
3. Padding, splitting, & merging
4. Detecting & replacing

Week 7: *Strings*

1. Making strings

2. Case conversion & substrings

3. Padding, splitting, & merging

4. Detecting & replacing

Install the `stringr` library

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

(Only do this once…and you already did this in HW 2)

Load the `stringr` library

```r
library(stringr)
```

(Do this every time you use the package)

Make a string with 'single' or "double" quotes

```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

Use double quotes when `'` is in the string

```r
cat("It's great!")
```

```
## It's great!
```

Use single quotes when `"` is in the string

```r
cat('I said, "Hello"')
```

```
## I said, "Hello"
```

What if a string has both ' and " symbols?

Example: `It's nice to say, "Hello"`

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

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

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

```
## 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, \"Hello\"") #
Double quote
```

```
## It's nice to say, "Hello"
```

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

```
## It's nice to say, "Hello"
```

Commonly escaped symbols:

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

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

```
cat('This\tthat') # Tab space: \t
```

```
## This	that
```

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

```
## This\that
```

"String constants": Sets of common strings

```
letters
```

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

```
LETTERS
```

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

"String constants": Sets of common strings

```
month.name
```

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

```
month.abb
```

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

The **stringr** library has a few *longer* string constants:

`fruit`, `words`, `sentences`

```r
length(fruit)
```

```
## [1] 80
```

```r
fruit[1:4]
```

```
## [1] "apple"   "apricot" "avocado" "banana"
```

```r
length(words)
```

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

```r
words[1:4]
```

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

```r
length(sentences)
```

```
## [1] 720
```

```r
sentences[1:4]
```

```
## [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."
```

Week 7: *Strings*

1. Making strings
2. Case conversion & substrings
3. Padding, splitting, & merging
4. Detecting & replacing

Case conversion & substrings

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_locate()`	returns indices of substrings
`str_dup()`	duplicates characters

Case conversion

```
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."
```

Comparing strings

Case matters:

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

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

Convert case *before* comparing if you just want to compare the string text:

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

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

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

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

Get the number of characters in a string

The `length()` function returns the *vector* length:

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

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

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

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

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

```
str_length(" ") # Spaces count
```

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

```
str_length("")  # Empty string
```

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

Access characters by their index with `str_sub()`

Indices start at 1:

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

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

Negative numbers count backwards from end:

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

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

Modify a string with `str_sub()`:

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

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

Get the indices of substrings

Extract the substring "Good" from the following string:

```
x <- 'thisIsGoodPractice'
```

1): Use str_locate() to get the **start** and **end** indices:

```
indices <- str_locate(x, 'Good')
indices
```

```
##      start end
## [1,]     7  10
```

2): Use str_sub() to get the substring:

```
str_sub(x, indices[1], indices[2])
```

```
## [1] "Good"
```

Repeat a string with str_dup()

```
str_dup("holla", 3)
```

```
## [1] "hollahollaholla"
```

Note the difference with rep():

```
rep("holla", 3)
```

```
## [1] "holla" "holla" "holla"
```

`stringr` functions work on vectors

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

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

Get the first 3 letters in each string:

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

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

Duplicate each string twice

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

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

Quick practice:

05:00

Create this string object: `x <- 'thisIsGoodPractice'`, then use **stringr** functions to transform `x` into the following strings:

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

Hint: You'll need these:

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

Week 7: *Strings*

1. Making strings
2. Case conversion & substrings
3. Padding, splitting, & merging
4. Detecting & replacing

Padding, splitting, & merging

Function	Description
`str_trim()`	removes leading and trailing whitespace
`str_pad()`	pads a string
`paste()`	string concatenation
`str_split()`	split a string into a vector

Remove excess white space with `str_trim()`

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

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

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

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

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

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

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

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

Add white space (or other characters) with `str_pad()`

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

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

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

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

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

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

```r
str_pad(x, width = 10, side = "both", pad = '*') # Specify the pad
```

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

Combine strings into one string with `paste()`

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

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

Control separation with `sep` argument (default is `" "`):

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

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

Note the difference with *vectors* of strings:

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

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

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

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

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

Split a string into multiple strings with `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"
## [3] "has"               "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 Example:

```r
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:

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

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

Common splits (memorize these)

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

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

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

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

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

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

Quick practice:

05:00

Create the following objects:

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

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: You'll need these:

- `str_trim()`
- `str_pad()`
- `paste()`
- `str_split()`

Think-Pair-Share

15:00

1) `reverseString(s)`: Write a function that returns the string `s` in reverse order.

- `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.

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

Break

05:00

Week 7: *Strings*

1. Making strings
2. Case conversion & substrings
3. Padding, splitting, & merging
4. Detecting & replacing

Detecting & replacing

Function	Description
`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

Sort string vectors alphabetically with `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"
```

Detect pattern in string: `str_detect(string, pattern)`

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

```
##  [1] "apple"        "apricot"      "avocado"
##  [4] "banana"       "bell pepper"  "bilberry"
##  [7] "blackberry"   "blackcurrant" "blood orange"
## [10] "blueberry"
```

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

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

How many in vector have the string `"berry"`?

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

Count number of times pattern appears in string

`str_count(string, pattern)`

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

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

Note the difference with `str_detect()`:

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

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

Detect if string *starts* with pattern

Which fruits *start* with "a"?

```
fiveFruit <- fruit[1:5]
fiveFruit
```

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

Wrong:

```
str_detect(fiveFruit, "a")
```

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

Right:

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

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

Detect if string *ends* with pattern

Which fruits *end* with an "e"?

```
fiveFruit
```

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

Wrong:

```
str_detect(fiveFruit, "e")
```

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

Right:

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

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

Remember:

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

```
fiveFruit
```

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

```
str_detect(fiveFruit, "^a") # Start with power (^)
```

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

```
str_detect(fiveFruit, "e$") # End with money ($)
```

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

Quick practice:

05:00

```r
fruit[1:5]
```

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

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

1. How many fruit have the string "rr" in it?
2. Which fruit end with string "fruit"?
3. Which fruit contain more than one "o" character?

Hint: You'll need to use str_detect() and str_count()

Replace matched strings with new string

`str_replace(string, pattern, replacement)`

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

```
str_replace(x, "a", "-") # Only replaces the first match
```

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

```
str_replace_all(x, "a", "-") # Replaces all matches
```

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

Quick practice redux

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

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

We did this earlier:

```
paste(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"
```

Think-Pair-Share

15:00

1) `sortString(s)`: Write the function `sortString(s)` that takes a string `s` and returns back an alphabetically sorted string.

- `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**.

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

Homeworks

- This is the last week to submit homeworks 1 - 6

Midterm

- Midterm is during class period next week, Oct. 20.
- Questions will be emailed as a PDF, type responses in .R file.

Midterm Review

- Planning on holding a one-hour review this week (tentatively Thursday).