# hw06-Tang-Jiahui

*Jiahui Tang*

*2017/11/3*

```
knitr::opts_chunk$set(warning = F)
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Warning: package 'purrr' was built under R version 3.4.2
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
library(stringr)
```

# Character data

Read and work the exercises in the Strings chapter (http://r4ds.had.co.nz/strings.html) or R for Data Science.

## Exercises14.2.5

- In code that doesn't use stringr, you'll often see `paste()` and `paste0()`. What's the difference between the two functions? What stringr function are they equivalent to? How do the functions differ in their handling of NA?

*Answer: I believe `paste0(..., collapse)` is equivalent to `paste(..., sep = "", collapse)`. We can use `str_c()` in stringr to replace them. However, `str_replace_na()` will be used when handling with NA if we use `str_c`, while `paste()` and `paste0()` won't need that. From the results below, we can find that `str_c()` returns NA if any arugment is NA while `paste()` and `paste0()` will replace NA with "NA".*

```
#Usage:
#paste (..., sep = " ", collapse = NULL)
#paste0(..., collapse = NULL)
x <- c(1:3,"ab","cd","ef", NA)
paste(x, sep = " ",collapse = NULL)
```

```
## [1] "1"  "2"  "3"  "ab" "cd" "ef" "NA"
```

```
str_c(str_replace_na(x), sep = " ")
```

```
## [1] "1"  "2"  "3"  "ab" "cd" "ef" "NA"
```

```
paste0(x, collapse = ",")
```

```
## [1] "1,2,3,ab,cd,ef,NA"
```

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

```
## [1] "1,2,3,ab,cd,ef,NA"
```

```
str_c("a",NA)
```

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

```
paste("a",NA)
```

```
## [1] "a NA"
```

```
paste0("a",NA)
```

```
## [1] "aNA"
```

- In your own words, describe the difference between the sep and collapse arguments to `str_c()`.

*Answer: `sep` is used to insert string between arguments of str_c, while `collapse` is the string used to combine input vectors into single string. See examples below.*

```
x <- c("aa","bb","cc")
str_c("Letter", x, sep = ", ")
```

```
## [1] "Letter, aa" "Letter, bb" "Letter, cc"
```

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

```
## [1] "aa, bb, cc"
```

- Use str_length() and str_sub() to extract the middle character from a string. What will you do if the string has an even number of characters?

*Answer: I would choose the str_length(x)/2 if the string has an even number of chars.*

```
x <- c("1","12","123","1234","12345","123456")
middle <- function(x) {
  if(str_length(x)%%2 == 0){
   res <- str_length(x)/2
  } else{
   res <- (str_length(x) + 1)/2
  }
  return(res)
}
mid <- middle(x)
str_sub(x, mid, mid)
```

```
## [1] "1" "1" "2" "2" "3" "3"
```

- What does str_wrap() do? When might you want to use it?

*Answer: Wrap strings into nicely formatted paragraphs. I think it is useful to handle long text like the example below.*

```
x <- c("What does str_wrap() do?
       When might you want to use it?")
cat(str_wrap(x,width = 10, exdent = 2), "\n")
```

```
## What does
##   str_wrap()
##   do? When
##   might you
##   want to
##   use it?
```

- What does str_trim() do? What's the opposite of str_trim()?

*Anwser: `str_trim()` is to delete whitespace from start and end of string. `str_pad` to add whitespace.*

```
str_trim(" What does str_trim() do? ")
```

```
## [1] "What does str_trim() do?"
```

```
x <- "What does str_trim() do?"
str_pad(x, str_length(x)+2, side = "both")
```

```
## [1] " What does str_trim() do? "
```

- Write a function that turns (e.g.) a vector c(“a”, “b”, “c”) into the string a, b, and c. Think carefully about what it should do if given a vector of length 0, 1, or 2.

```
vector_to_string <- function(x) {
  if(length(x) > 1){
    str_c(str_c(x[-length(x)], collapse = ", "), x[length(x)], sep = ", and ")
  } else{
    return(x)
  }
}
x <- c("a", "b", "c","d")
vector_to_string(x)
```

```
## [1] "a, b, c, and d"
```

# Exercises14.3.1.1

- Explain why each of these strings don’t match a : “",”\“,”\".

*Answer: From the results below, we can find that “" would escape the next character,”\" would become  in regexps, “\" works like the combination of”" and “\” as it would become a backblash then escape the next character.*

```
backslash <- "a\b"
backslash1 <- "a\\b"
backslash2 <- "a\\\b"
writeLines(backslash)
```

```
## a
```

```
writeLines(backslash1)
```

```
## a\b
```

```
writeLines(backslash2)
```

```
## a\
```

```
str_view(c("abc", "a\\b", "bef"), "\\\\")
```

abc

a\b

bef

- How would you match the sequence “’ ?

*Anwser: We should string “"\’\\” to match “’ as below.*

```
str_view(c("abc", 'a"\'\\b', "bef"), "\"\\'\\\\")
```

abc

a"'\b

bef

- What patterns will the regular expression ..... match? How would you represent it as a string?

*Answer: The regular express ..... matches any strings that have a dot followed by any characters,then another dot with character，then another dot with character.*

```
str_view(c("abc", ".a.b.c", "bef"), "\\..\\..\\..")
```

abc

.a.b.c

bef

# Exercises14.3.2.1

- How would you match the literal string "^"?

*Answer：We can use '^\\^\\$' to match.*

```
str_view(c("abc", "$^$", "a$^$a"), "^\\$\\^\\$$")
```

abc

$^$

a$^$a

- Given the corpus of common words in stringr::words, create regular expressions that find all words that:

1. Start with "y".
2. End with "x"
3. Are exactly three letters long. (Don't cheat by using str_length()!)
4. Have seven letters or more.

Since this list is long, you might want to use the match argument to str_view() to show only the matching or non-matching words.

*Anwser: We can use regular expressions show below to find all words.*

```
str_view(stringr::words, "^y", match = TRUE)
```

year

yes

yesterday

yet

you

young

```
str_view(stringr::words, "x$", match = TRUE)
```

box

sex

six

tax

```
str_view(tail(stringr::words), "^...$", match = TRUE)
```

yes

yet

you

```
str_view(head(stringr::words), "^.......", match = TRUE)
```

absolute

account

# Exercises14.3.3.1

- Create regular expressions to find all words that:

1. Start with a vowel.
2. That only contain consonants. (Hint: thinking about matching "not"-vowels.)
3. End with ed, but not with eed.
4. End with ing or ise.

```
str_view(head(stringr::words), "^[aeiou]", match = TRUE)
```

a

able

about

absolute

accept

account

```
str_view(stringr::words, "[aeiou]", match = FALSE)
```

by

dry

fly

mrs

try

why

```
str_view(stringr::words, "[^e]ed$", match = TRUE)
```

bed

hundred

red

```
str_view(stringr::words, "ing$|ise$", match = TRUE)
```

advertise

bring

during

evening

exercise

king

meaning

morning

otherwise

practise

raise

realise

ring

rise

sing

surprise

thing

- Empirically verify the rule “i before e except after c”.

```
str_view(stringr::words, "cei|[^c]ie", match = TRUE)
```

achieve

believe

brief

client

die

experience

field

friend

lie

piece

quiet

receive

tie

view

```
str_view(stringr::words, "cie|[^c]ei", match = TRUE)
```

science

society

weigh

- Is “q” always followed by a “u”?

*Answer: From the result we may find it is true that “q” always followed by a “u”.*

```
str_view(stringr::words, "q[^u]", match = TRUE)
```

- Write a regular expression that matches a word if it’s probably written in British English, not American English.

*Answer: Taking “-ize/-ise” and “-se/-ze” as examples, we may use regular expression to match British English words.*

```
str_view(c("summarize","summarise","defense","defence"), "ce$|ize$", match = TRUE)
```

summarize

defence

- Create a regular expression that will match telephone numbers as commonly written in your country.

```
str_view(c("778-111-1111","028-8888-8888"), "\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d", match = TRUE)
```

778-111-1111

# Exercises14.3.4.1

- Describe the equivalents of ?, +, * in {m,n} form.

*Answer: ? is equal to {,1}, + is equal to {1,} and * is equal to {0,}*

- Describe in words what these regular expressions match: (read carefully to see if I’m using a regular expression or a string that defines a regular expression.)

1. ^.*$
2. “\{.+\}”
3. --
4. “\\{4}”

*Anwser: The first regular expression matches all words. The second one matches all words(at least one character) surrounded by curly braces. The third one matches XXXX-XX-XX(X represents digits). The last one matches four backslashes.*

```
str_view(stringr::words, "^.*$", match = FALSE)
```

```
str_view(c("{a}","a{aaaa}","aa"), "\\{.+\\}", match = TRUE)
```

{a}

a{aaaa}

```
str_view(c("1234-56-78","a{a}","aa"), "\\d{4}-\\d{2}-\\d{2}", match = TRUE)
```

1234-56-78

```
str_view(c("\\\\\\\\","a{a}","aa"), "\\\\{4}", match = TRUE)
```

\\\\

- Create regular expressions to find all words that:

1. Start with three consonants.
2. Have three or more vowels in a row.
3. Have two or more vowel-consonant pairs in a row.

```
str_view(stringr::words, "^[^aeoiu]{3,}", match = TRUE)
```

Christ

Christmas

dry

fly

mrs

scheme

school

straight

strategy

street

strike

strong

structure

system

three

through

throw

try

type

why

```
str_view(stringr::words, "[aeoiu]{3,}", match = TRUE)
```

beauty

obvious

previous

quiet

serious

various

```
str_view(head(stringr::words), "([aeoui][^aeoiu]){2,}", match = TRUE)
```

absolute

- Solve the beginner regexp crosswords at https://regexcrossword.com/challenges/beginner (https://regexcrossword.com/challenges/beginner).

*Anwser: help bobe oooo **// 1984*

# Exercises14.3.5.1

- Describe, in words, what these expressions will match:

1. (.)
2. “(.)(.)\2\1”
3. (..)
4. “(.).\1.\1”
5. “(.)(.)(.).*\3\2\1"

*Answer: The first one means the same characters showing three times. The second one represents a pair of characters and its reversal. The third one matches two charcaters followed by the same two characters. The forth matches a character followed by any character, then character same as the first, then any other character and the same character again. The last one matches three characters followed by 0 or more characters, the the three characters in reverse.*

```
str_view(c("aa","aaa"), "(.)\\1\\1", match = TRUE)
```

aaa

```
str_view(c("abba","abab"), "(.)(.)\\2\\1", match = TRUE)
```

abba

```
str_view(c("abcd","aaaa","a0a0"), "(..)\\1", match = TRUE)
```

aaaa

a0a0

```
str_view(c("a1a2a","a.a.a","abcde"), "(.).\\1.\\1", match = TRUE)
```

a1a2a

a.a.a

```
str_view(c("abc123cba","abcderf"), "(.)(.)(.).*\\3\\2\\1", match = TRUE)
```

abc123cba

- Construct regular expressions to match words that:

1. Start and end with the same character.
2. Contain a repeated pair of letters (e.g. “church” contains “ch” repeated twice.)
3. Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.)

```
str_view(c("abc123cba","abcderf"), "^(.).*\\1$", match = TRUE)
```

abc123cba

```
str_view(c("abcab","abcderf","cdabcab"), ".*(..).*\\1", match = TRUE)
```

abcab

cdabcab

```
str_view(c("12311","abcaa","abcde"), ".*(.).*\\1.*\\1", match = TRUE)
```

12311

abcaa

# Exercises14.4.2

- For each of the following challenges, try solving it by using both a single regular expression, and a combination of multiple str_detect() calls.

1. Find all words that start or end with x.
2. Find all words that start with a vowel and end with a consonant.
3. Are there any words that contain at least one of each different vowel?

```
#1.1 A sigle regexp
words[str_detect(words,"^x|x$")]
```

```
## [1] "box" "sex" "six" "tax"
```

```
#1.2 Combination of mutiple str_detect() calls
start_x <- str_detect(words,"^x")
end_x <- str_detect(words,"x$")
words[start_x|end_x]
```

```
## [1] "box" "sex" "six" "tax"
```

```
#2.1 A sigle regexp
head(str_subset(words, "^[aeiou].*[^aeiou]$"))
```

```
## [1] "about"   "accept"  "account" "across"  "act"     "actual"
```

```
#2.2 Combination of mutiple str_detect() calls
start_vowel <- str_detect(words,"^[aeiou]")
end_consonant <- str_detect(words,"[^aeiou]$")
head(words[start_vowel&end_consonant])
```

```
## [1] "about"   "accept"  "account" "across"  "act"     "actual"
```

```
#3.1 A sigle regexp
str_subset(words, "[^a]&[^e]&[^i]&[^o]&[^u]")
```

```
## character(0)
```

```
#3.2 Str_detect()
words[str_detect(words,"a")&str_detect(words,"b")&str_detect(words,"c")&str_detect(words,"d")&str_detect(words,"e
")]
```

```
## character(0)
```

- What word has the highest number of vowels? What word has the highest proportion of vowels? (Hint: what is the denominator?)

*Anwser: From the result we can see “appropriate” “associate” “available” “colleague” “encourage” “experience” “individual” “television” have the highest number of vowels. For the second question, we can use str_length(words) as the denominator.*

```
#highest number of vowels
max(str_count(words, "[aeiou]"))
```

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

```
words[which(str_count(words, "[aeiou]") == 5)]
```

```
## [1] "appropriate" "associate"   "available"   "colleague"   "encourage"
## [6] "experience"  "individual"  "television"
```

```
#highest proportion of vowels
highest_vowels <- str_count(words, "[aeiou]")/str_length(words)
words[which(highest_vowels == max(highest_vowels))]
```

```
## [1] "a"
```

# Exercises14.4.3.1

- In the previous example, you might have noticed that the regular expression matched “flickered”, which is not a colour. Modify the regex to fix the problem.

*Anwser: We can add to solve this proble, like `str_c("\\b(", str_c(colours, collapse = "|"), ")\\b")`.*

- From the Harvard sentences data, extract:

1. The first word from each sentence.
2. All words ending in ing.
3. All plurals.

*Anwser: For question3, we just consider the situation that words with more than 3 characters ending with “s”. From the result, we may find that there are some false postives like “makes”.*

```
# First word
head(str_extract(sentences,"[a-zA-Z]+"))
```

```
## [1] "The"   "Glue"  "It"    "These" "Rice"  "The"
```

```
# All words ending in ing
ing <- str_extract_all(sentences[str_detect(sentences, "\\b[a-zA-Z]+ing\\b")],"\\b[a-zA-Z]+ing\\b",simplify = TRU
E)
head(ing)
```

```
##      [,1]
## [1,] "spring"
## [2,] "evening"
## [3,] "morning"
## [4,] "winding"
## [5,] "living"
## [6,] "king"
```

```
# All plurals
plurals <- str_extract_all(sentences[str_detect(sentences, "\\b[a-zA-Z]{3,}s\\b")],"\\b[a-zA-Z]{3,}s\\b",simplify
= TRUE)
head(plurals)
```

```
##      [,1]     [,2]    [,3]
## [1,] "planks" ""      ""
## [2,] "days"   ""      ""
## [3,] "bowls"  ""      ""
## [4,] "lemons" "makes" ""
## [5,] "hogs"   ""      ""
## [6,] "hours"  ""      ""
```

# Exercises14.4.4.1

- Find all words that come after a “number” like “one”, “two”, “three” etc. Pull out both the number and the word.

```
number <-"(one|two|three|four|five|six|seven|eight|nine|ten) ([^ ]+)"
after_number <- sentences %>%
  str_subset(number) %>%
  head(10)
after_number %>%
  str_extract(number)
```

```
##  [1] "ten served"  "one over"    "seven books" "two met"     "two factors"
##  [6] "one and"     "three lists" "seven is"    "two when"    "one floor."
```

- Find all contractions. Separate out the pieces before and after the apostrophe.

```
contraction <- "([A-Za-z]+)'([A-Za-z]+)"
con <- sentences %>%
  str_subset(contraction) %>%
  head(10)
con %>%
  str_match(contraction)
```

```
##       [,1]        [,2]      [,3]
##  [1,] "It's"      "It"      "s"
##  [2,] "man's"     "man"     "s"
##  [3,] "don't"     "don"     "t"
##  [4,] "store's"   "store"   "s"
##  [5,] "workmen's" "workmen" "s"
##  [6,] "Let's"     "Let"     "s"
##  [7,] "sun's"     "sun"     "s"
##  [8,] "child's"   "child"   "s"
##  [9,] "king's"    "king"    "s"
## [10,] "It's"      "It"      "s"
```

# Exercises14.4.5.1

- Replace all forward slashes in a string with backslashes.

```
x <- c("/apple", "pe/ar", "bana/na")
str_replace(x, "[////]", "\\\\")
```

```
## [1] "\\apple"  "pe\\ar"   "bana\\na"
```

```
writeLines(x)
```

```
## /apple
## pe/ar
## bana/na
```

- Implement a simple version of str_to_lower() using replace_all().

```
x <- c("housE", "carS", "PeoplE")
str_replace_all(x, c("E" = "e", "S" = "s", "P" = "p"))
```

```
## [1] "house"  "cars"   "people"
```

- Switch the first and last letters in words. Which of those strings are still words?

```
switch <- str_replace(words,"(^[a-zA-Z])([a-zA-Z]*)([a-zA-Z]$)", "\\3\\2\\1")
head(switch)
```

```
## [1] "a"        "ebla"     "tboua"    "ebsoluta" "tccepa"   "tccouna"
```

```
words[words==switch]
```

```
##  [1] "a"         "america"   "area"      "dad"       "dead"
##  [6] "depend"    "educate"   "else"      "encourage" "engine"
## [11] "europe"    "evidence"  "example"   "excuse"    "exercise"
## [16] "expense"   "experience" "eye"      "health"    "high"
## [21] "knock"     "level"     "local"     "nation"    "non"
## [26] "rather"    "refer"     "remember"  "serious"   "stairs"
## [31] "test"      "tonight"   "transport" "treat"     "trust"
## [36] "window"    "yesterday"
```

# Exercises14.4.6.1

- Split up a string like “apples, pears, and bananas” into individual components.

```
x <- c("apples, pears, and bananas")
str_split(x, boundary("word"))[[1]]
```

```
## [1] "apples"  "pears"   "and"     "bananas"
```

- Why is it better to split up by boundary(“word”) than " “?

*Answer: It is better to split up by boundary(“word”) because there are also punctuation and not just whitespace.*

- What does splitting with an empty string (“”) do? Experiment, and then read the documentation.

*Answer: string“” will split a string into single characters including punctuations and whitespaces*

```
str_split("ab.cd. e ", "")[[1]]
```

```
## [1] "a" "b" "." "c" "d" "." " " "e" " "
```

# Exercises14.5.1

- How would you find all strings containing  with regex() vs. with fixed()?

```
str_subset(c("a\\b", "ab"), "\\\\")
```

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

```
str_subset(c("a\\b", "ab"), fixed("\\"))
```

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

- What are the five most common words in sentences?

```
str_extract_all(sentences, boundary("word")) %>%
  unlist() %>%
  str_to_lower() %>%
  tibble() %>%
  set_names("words") %>%
  group_by(words) %>%
  count(sort = TRUE) %>%
  head(5)
```

```
## # A tibble: 5 x 2
## # Groups:   words [5]
##   words     n
##   <chr> <int>
## 1   the   751
## 2     a   202
## 3    of   132
## 4    to   123
## 5   and   118
```

# Exercises14.7.1

- Find the stringi functions that:

1. Count the number of words.
2. Find duplicated strings.
3. Generate random text.

*Answer: We can use `stri_count_words` to count the number of words. `stri_duplicated` and `stri_duplicated_any` can be used to find duplicated strings. We can use stri_rand_strings to generate Random Strings.*

```
stringi::stri_count_words("words and sentences")
```

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

```
stringi::stri_duplicated(c("a", "b", "a", NA, "a", NA), fromLast=TRUE)
```

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

```
stringi::stri_duplicated_any(c("a", "b", "a", NA, "a", NA))
```

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

```
stringi::stri_rand_strings(2, 5)
```

```
## [1] "Hw2Yy" "ijl0q"
```

- How do you control the language that stri_sort() uses for sorting?

*Answer: We can use `opts_collator` arugment in `stri_sort(str, decreasing = FALSE, na_last = NA, ...,opts_collator = NULL)`.*

```
stringi::stri_sort(c("hladny", "chladny"), locale="pl_PL")
```

```
## [1] "chladny" "hladny"
```