Assignment 8

Adrian Bracher (Matr. Nr. 01637180)

19.05.2021

Contents

1 String basics

1.1 Quotes

There is different syntax for string literals, check it out below:

```
# Define line1
line1 <- "The table was a large one, but the three were all crowded together at one corner of it:"

# Define line2
line2 <- '"No room! No room!" they cried out when they saw Alice coming.'

# Define line3
line3 <- "\"There's plenty of room!\" said Alice indignantly, and she sat down in a large arm-chair at
```

1.2 What you see isn't always what you have

The function writeLines() concatenates strings and prints them.

```
# Putting lines in a vector
lines <- c(line1, line2, line3)

# Print lines
print(lines)

# Use writeLines() on lines
writeLines(lines)

# Write lines with a space separator
writeLines(lines, sep=" ")

# Use writeLines() on the string "hello\n\U1F30D"
writeLines("hello\n\U1F30D")
```

1.3 Escape sequences

If you need special characters in the string literals you need to escape them with a .

```
# Should display: To have a \ you need \\
writeLines("To have a \\ you need \\\\")

# Should display:
```

```r
# This is a really
# really really
# long string
writeLines("This is a really really really long string")

# Use writeLines() with
# "\u0928\u092e\u0938\u094d\u0924\u0947 \u0926\u0941\u0928\u093f\u092f\u093e"
writeLines("\u0928\u092e\u0938\u094d\u0924\u0947 \u0926\u0941\u0928\u093f\u092f\u093e")
```

1.4 Using format() with numbers

The function format() is used to format floats and specify how many digits the numbers should be formatted to. If scientific = false is passed to format(), then all the numbers of the vector are formatted to the same fixed amount of characters.

```r
# Some vectors of numbers
percent_change  <- c(4, -1.91, 3.00, -5.002)
income <- c(72.19, 1030.18, 10291.93, 1189192.18)
p_values <- c(0.12, 0.98, 0.0000191, 0.00000000002)

# Format c(0.0011, 0.011, 1) with digits = 1
format(c(0.0011, 0.011, 1), digits=1)
```

```
## [1] "0.001" "0.011" "1.000"
```

```r
# Format c(1.0011, 2.011, 1) with digits = 1
format(c(1.0011, 2.011, 1), digits=1)
```

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

```r
# Format percent_change to one place after the decimal point
format(percent_change, digits=2)
```

```
## [1] " 4.0" "-1.9" " 3.0" "-5.0"
```

```r
# Format income to whole numbers
format(income, scientific=FALSE, digits = 2)
```

```
## [1] "     72" "   1030" "  10292" "1189192"
```

```r
# Format p_values in fixed format
format(p_values, scientific=FALSE)
```

```
## [1] "0.12000000000" "0.98000000000" "0.00001910000" "0.00000000002"
```

1.5 Controlling other aspects of the string

In this exercise we use what we learned in the previous exercise and also introduce the parameter trim to remove extra spaces.

```r
formatted_income <- format(income, digits = 2)

# Print formatted_income
formatted_income
```

```
## [1] "     72" "   1030" "  10292" "1189192"
```

```r
# Call writeLines() on the formatted income
writeLines(formatted_income)
```

```
##      72
##    1030
##   10292
## 1189192
```

```
# Define trimmed_income
trimmed_income = format(income, digits=2, trim=TRUE)

# Call writeLines() on the trimmed_income
writeLines(trimmed_income)
```

```
## 72
## 1030
## 10292
## 1189192
```

```
# Define pretty_income
pretty_income = format(income, digits=2, big.mark=",")

# Call writeLines() on the pretty_income
writeLines(pretty_income)
```

```
##        72
##     1,030
##    10,292
## 1,189,192
```

1.6 formatC()

The function formatC() is an alternative to format which is inspired by C style syntax.

```
# From the format() exercise
x <- c(0.0011, 0.011, 1)
y <- c(1.0011, 2.011, 1)

# formatC() on x with format = "f", digits = 1
formatC(x, format = "f", digits = 1)
```

```
## [1] "0.0" "0.0" "1.0"
```

```
# formatC() on y with format = "f", digits = 1
formatC(y, format = "f", digits = 1)
```

```
## [1] "1.0" "2.0" "1.0"
```

```
# Format percent_change to one place after the decimal point
formatC(percent_change, format = "f", digits = 1)
```

```
## [1] "4.0"  "-1.9" "3.0"  "-5.0"
```

```
# percent_change with flag = "+"
formatC(percent_change, format = "f", digits = 1, flag = "+")
```

```
## [1] "+4.0" "-1.9" "+3.0" "-5.0"
```

```
# Format p_values using format = "g" and digits = 2
formatC(p_values, format = "g", digits = 2)# From the format() exercise
```

```
## [1] "0.12"    "0.98"    "1.9e-05" "2e-11"
```

```
x <- c(0.0011, 0.011, 1)
y <- c(1.0011, 2.011, 1)

# formatC() on x with format = "f", digits = 1
formatC(x, format = "f", digits = 1)
```

```
## [1] "0.0" "0.0" "1.0"
```

```
# formatC() on y with format = "f", digits = 1
formatC(y, format = "f", digits = 1)
```

```
## [1] "1.0" "2.0" "1.0"
```

```
# Format percent_change to one place after the decimal point
formatC(percent_change, format = "f", digits = 1)
```

```
## [1] "4.0"  "-1.9" "3.0"  "-5.0"
```

```
# percent_change with flag = "+"
formatC(percent_change, format = "f", digits = 1, flag = "+")
```

```
## [1] "+4.0" "-1.9" "+3.0" "-5.0"
```

```
# Format p_values using format = "g" and digits = 2
formatC(p_values, format = "g", digits = 2)
```

```
## [1] "0.12"    "0.98"    "1.9e-05" "2e-11"
```

1.7 Annotation of numbers

Paste is used to format vectors of number strings. In the example below we use it to add characters in an automated way and also collapse the vector into one string.

```
# Add $ to pretty_income
paste("$", pretty_income, sep = "")

# Add % to pretty_percent
paste(pretty_percent, "%", sep = "")

# Create vector with elements like 2010: +4.0%`
year_percent <-paste(years, ": ", pretty_percent, "%", sep = "")

# Collapse all years into single string
paste(year_percent, collapse=", ")
```

1.8 A very simple table

In the following exercise we combine what we've learned to format two vectors neatly in a table.

```
# Define the names vector
income_names <- c("Year 0", "Year 1", "Year 2", "Project Lifetime")

# Create pretty_income
pretty_income <- format(income, digits = 2, big.mark = ",")

# Create dollar_income
dollar_income <- paste("$", pretty_income, sep="")
```

```r
# Create formatted_names
formatted_names <- format(income_names, justify="right")

# Create rows
rows = paste(formatted_names, dollar_income, sep="   ")

# Write rows
writeLines(rows)
```

```
##           Year 0   $       72
##           Year 1   $    1,030
##           Year 2   $   10,292
## Project Lifetime   $1,189,192
```

1.9 Let's order pizza

Again, in this exercise the previously introduced functions are used to format string vectors.

```r
# Randomly sample 3 toppings
my_toppings <- sample(toppings, size = 3)

# Print my_toppings
my_toppings

# Paste "and " to last element: my_toppings_and
my_toppings_and <- paste(c("", "", "and "), my_toppings, sep="")

# Collapse with comma space: these_toppings
these_toppings <- paste(my_toppings_and, collapse=", ")

# Add rest of sentence: my_order
my_order <- paste("I want to order a pizza with ", these_toppings, ".", sep="")

# Order pizza with writeLines()
writeLines(my_order)
```

2 Introduction

2.1 Introduction to stringr

In this exercise the stringr function str_c is compared to paste().

```r
library(stringr)

my_toppings <- c("cheese", NA, NA)
my_toppings_and <- paste(c("", "", "and "), my_toppings, sep = "")

# Print my_toppings_and
my_toppings_and
```

```
## [1] "cheese" "NA"     "and NA"
```

```r
# Use str_c() instead of paste(): my_toppings_str
my_toppings_str <- str_c(c("", "", "and "), my_toppings, sep="")
```

```
# Print my_toppings_str
my_toppings_str
```

```
## [1] "cheese" NA       NA
```

```
# paste() my_toppings_and with collapse = ", "
paste(my_toppings_and, collapse=", ")
```

```
## [1] "cheese, NA, and NA"
```

```
# str_c() my_toppings_str with collapse = ", "
str_c(my_toppings_str, collapse=", ")
```

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

2.2 String length

The function str_length is introduces, which returns the length of the individual strings of a vector.

```
library(stringr)
library(babynames)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Extracting vectors for boys' and girls' names
babynames_2014 <- filter(babynames, year == 2014)
boy_names <- filter(babynames_2014, sex == "M")$name
girl_names <- filter(babynames_2014, sex == "F")$name

# Take a look at a few boy_names
head(boy_names)
```

```
## [1] "Noah"    "Liam"    "Mason"   "Jacob"   "William" "Ethan"
```

```
# Find the length of all boy_names
boy_length <- str_length(boy_names)

# Take a look at a few lengths
head(boy_length)
```

```
## [1] 4 4 5 5 7 5
```

```
# Find the length of all girl_names
girl_length <- str_length(girl_names)

# Find the difference in mean length
mean(girl_length) - mean(boy_length)
```

```
## [1] 0.3374758
```

```
# Confirm str_length() works with factors
head(str_length(factor(boy_names)))
```

```
## [1] 4 4 5 5 7 5
```

2.3 Extracting substrings

In this exercise we learn how to use str_sub to get substrings of a string vector. The second and third parameters specify the beginning and end of the desired substrings.

```
# Extract first letter from boy_names
boy_first_letter <- str_sub(boy_names, 1, 1)

# Tabulate occurrences of boy_first_letter
table(boy_first_letter)
```

```
## boy_first_letter
##    A    B    C    D    E    F    G    H    I    J    K    L    M    N    O    P 
## 1454  651  770  998  549  185  334  403  235 1390 1291  537  914  424  207  230 
##    Q    R    S    T    U    V    W    X    Y    Z 
##   56  778  806  771   43  160  174   56  252  379
```

```
# Extract the last letter in boy_names, then tabulate
boy_last_letter <- str_sub(boy_names, -1, -1)
table(boy_last_letter)
```

```
## boy_last_letter
##    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p 
##  421  104   92  436 1148   66   82  583  705   57  349  945  389 4672  730   32 
##    q    r    s    t    u    v    w    x    y    z 
##   19 1011  826  292   81   71   34   86  697  119
```

```
# Extract the first letter in girl_names, then tabulate
girl_first_letter <- str_sub(girl_names, 1, 1)
table(girl_first_letter)
```

```
## girl_first_letter
##    A    B    C    D    E    F    G    H    I    J    K    L    M    N    O    P 
## 3101  699  946  810  933  209  345  469  373 1430 1694 1122 1746  752  143  303 
##    Q    R    S    T    U    V    W    X    Y    Z 
##   38  831 1369  683   28  214   85   62  294  502
```

```
# Extract the last letter in girl_names, then tabulate
girl_last_letter <- str_sub(girl_names, -1, -1)
table(girl_last_letter)
```

```
## girl_last_letter
##    a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p 
## 6632   20   13   81 3114    8   21 1942 1581   12   31  450  115 2608  105    3 
##    q    r    s    t    u    v    w    x    y    z 
##    2  291  326  208   59    6   17   50 1435   51
```

2.4 Detecting matches

In this exercise we use str_detect to detect occurrences of specified patterns.

```
# Look for pattern "zz" in boy_names
contains_zz <- str_detect(boy_names, "zz")
```

```r
# Examine str() of contains_zz
str(contains_zz)

# How many names contain "zz"?
sum(contains_zz)

# Which names contain "zz"?
boy_names[contains_zz]

# Which rows in boy_df have names that contain "zz"?
boy_df[contains_zz, ]
```

2.5 Subsetting strings based on match

Finding strings that contain a certain pattern is often necessary, so stringr has a function that does it in one step called str_subset.

```r
# Find boy_names that contain "zz"
str_subset(boy_names, "zz")
```

```
##  [1] "Uzziah"    "Ozzie"     "Ozzy"      "Jazz"      "Uzziel"    "Chazz"
##  [7] "Izzy"      "Azzam"     "Izzac"     "Izzak"     "Fabrizzio" "Jazziel"
## [13] "Azzan"     "Izzaiah"   "Muizz"     "Yazziel"
```

```r
# Find girl_names that contain "zz"
str_subset(girl_names, "zz")
```

```
##  [1] "Izzabella"  "Jazzlyn"    "Jazzlynn"   "Lizzie"     "Izzy"
##  [6] "Lizzy"      "Mazzy"      "Izzabelle"  "Jazzmine"   "Jazzmyn"
## [11] "Jazzelle"   "Jazzmin"    "Izzah"      "Jazzalyn"   "Jazzmyne"
## [16] "Izzabell"   "Jazz"       "Mazzie"     "Alyzza"     "Izza"
## [21] "Izzie"      "Jazzlene"   "Lizzeth"    "Jazzalynn"  "Jazzy"
## [26] "Alizzon"    "Elizzabeth" "Jazzilyn"   "Jazzlynne"  "Jizzelle"
## [31] "Izzabel"    "Izzabellah" "Izzibella"  "Jazzabella" "Jazzabelle"
## [36] "Jazzel"     "Jazzie"     "Jazzlin"    "Jazzlyne"   "Aizza"
## [41] "Brizza"     "Ezzah"      "Fizza"      "Izzybella"  "Rozzlyn"
```

```r
# Find girl_names that contain "U"
starts_U <- str_subset(girl_names, "U")
starts_U
```

```
##  [1] "Unique"  "Uma"     "Unknown" "Una"     "Uriah"   "Ursula"  "Unity"
##  [8] "Umaiza"  "Urvi"    "Ulyana"  "Ula"     "Udy"     "Urwa"    "Ulani"
## [15] "Umaima"  "Umme"    "Ugochi"  "Ulyssa"  "Umika"   "Uriyah"  "Ubah"
## [22] "Umaira"  "Umi"     "Ume"     "Urenna"  "Uriel"   "Urijah"  "Uyen"
```

```r
# Find girl_names that contain "U" and "z"
str_subset(starts_U, "z")
```

```
## [1] "Umaiza"
```

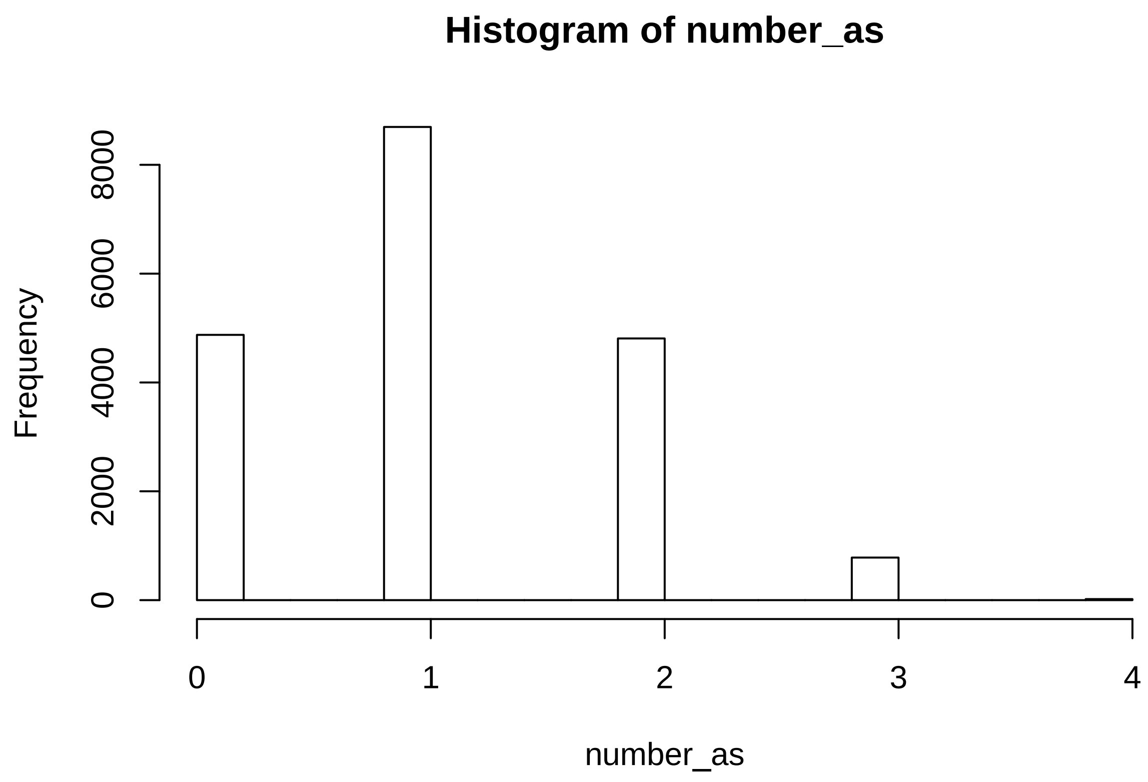
2.6 Counting matches

Another useful function is str_count, which counts the occurrences of a pattern in a given string.

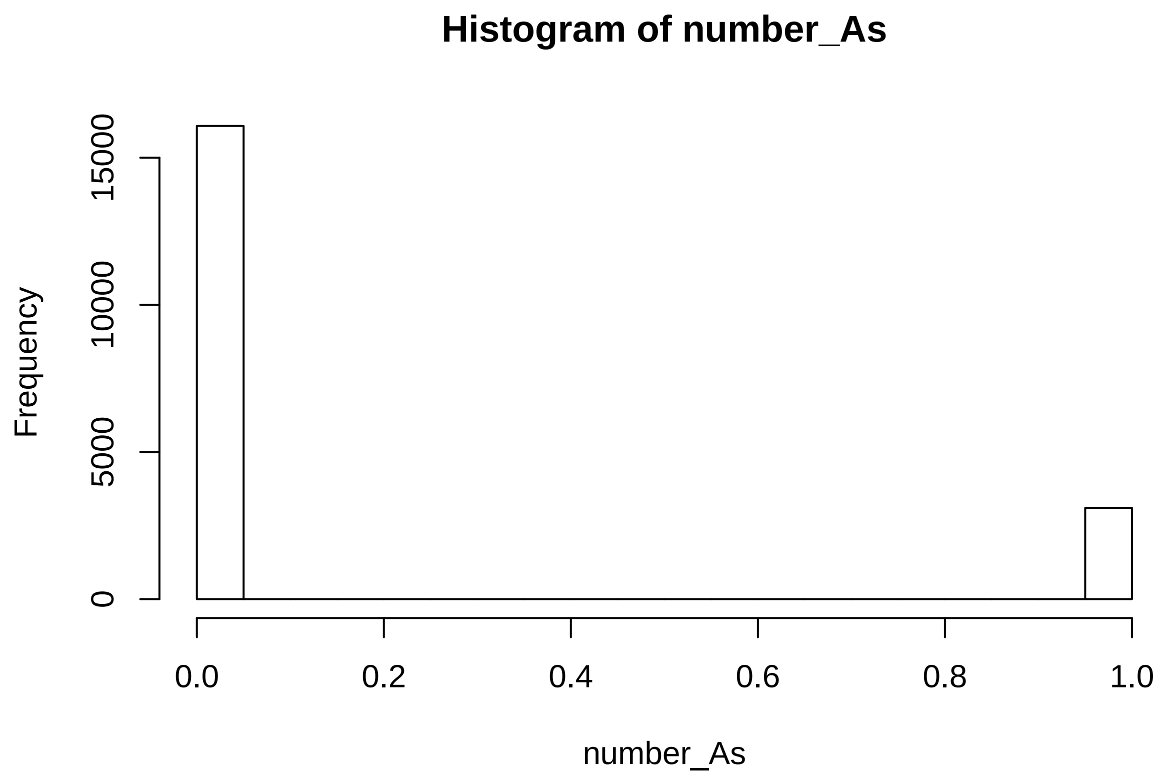
```r
# Count occurrences of "a" in girl_names
number_as <- str_count(girl_names, "a")
```

```r
# Count occurrences of "A" in girl_names
number_As <- str_count(girl_names, "A")

# Histograms of number_as and number_As
hist(number_as)
```



```r
hist(number_As)
```



```
# Find total "a" + "A"
total_as <- number_as + number_As

# girl_names with more than 4 a's
girl_names[total_as > 4]
```

```
## [1] "Aaradhana"
```

2.7 Parsing strings into variables

In the following we exercise the use of str_split.

```
# From previous step
date_ranges <- c("23.01.2017 - 29.01.2017", "30.01.2017 - 06.02.2017")
split_dates_n <- str_split(date_ranges, fixed(" - "), n = 2, simplify = TRUE)

# Subset split_dates_n into start_dates and end_dates
start_dates <- split_dates_n[,1]

# Split start_dates into day, month and year pieces
str_split(start_dates, fixed("."), n = 3, simplify = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,] "23" "01" "2017"
## [2,] "30" "01" "2017"
```

```
both_names <- c("Box, George", "Cox, David")
```

```r
# Split both_names into first_names and last_names
both_names_split <- str_split(both_names, fixed(", "), n = 2, simplify = TRUE)

# Get first names
first_names <- both_names_split[, 2]

# Get last names
last_names <- both_names_split[, 1]
```

2.8 Some simple text statistics

Here we add lapply to the mix to figure out the lengths for each element in the vectors.

```r
# Split lines into words
words <- str_split(lines, " ")

# Number of words per line
lapply(words, length)

# Number of characters in each word
word_lengths <- lapply(words, str_length)

# Average word length per line
lapply(word_lengths, mean)
```

2.9 Replacing to tidy strings

In the following we go through examples of how to use str_replace. All occurrences (or just the first for str_replace) of the second parameter are replaced by third parameter.

```r
# Some IDs
ids <- c("ID#: 192", "ID#: 118", "ID#: 001")

# Replace "ID#: " with ""
id_nums <- str_replace(ids, "ID#: ", "")

# Turn id_nums into numbers
id_ints <- as.numeric(id_nums)

# Some (fake) phone numbers
phone_numbers <- c("510-555-0123", "541-555-0167")

# Use str_replace() to replace "-" with " "
str_replace(phone_numbers, "-", " ")
```

```
## [1] "510 555-0123" "541 555-0167"
```

```r
# Use str_replace_all() to replace "-" with " "
str_replace_all(phone_numbers, "-", " ")
```

```
## [1] "510 555 0123" "541 555 0167"
```

```r
# Turn phone numbers into the format xxx.xxx.xxxx
str_replace_all(phone_numbers, "-", ".")
```

```
## [1] "510.555.0123" "541.555.0167"
```

2.10 Review

This exercise revisits all that we've learned in this chapter.

```r
genes = readRDS("dna.rds")
# Find the number of nucleotides in each sequence
str_length(genes)
```

```
## [1] 441 462 993
```

```r
# Find the number of A's occur in each sequence
str_count(genes, fixed("A"))
```

```
## [1] 118 117 267
```

```r
# Return the sequences that contain "TTTTTT"
str_subset(genes, fixed("TTTTTT"))
```

```
## [1] "TTAAGGAACGATCGTACGCATGATAGGGTTTTGCAGTGATATTAGTGTCTCGGTTGACTGGATCTCATCAATAGTCTGGATTTTGTTGATAAGTA
```

```r
# Replace all the "A"s in the sequences with a "_"
str_replace_all(genes, fixed("A"), "_")
```

```
## [1] "TT_G_GT___TT__TCC__TCTTTG_CCC___TCTCTGCTGG_TCCTCTGGT_TTTC_TGTTGG_TG_CGTC__TTTCT__T_TTTC_CCC__CC
## [2] "TT__GG__CG_TCGT_CGC_TG_T_GGGTTTTGC_GTG_T_TT_GTGTCTCGGTTG_CTGG_TCTC_TC__T_GTCTGG_TTTTGTTG_T__GT_
## [3] "_TG______C__TTT_TCC_____C__C__C___TC_GCTTCGT____TC_TTCTTTTCCCGCC__TT_G_GC__C__CTTGGCTTG_TCG__GT
```

2.11 Final challenges

Again, we deepen the learned knowledge in this exercise.

```r
# Define some full names
names <- c("Diana Prince", "Clark Kent")

# Split into first and last names
names_split <- str_split(names, fixed(" "), simplify = TRUE)

# Extract the first letter in the first name
abb_first <- str_sub(names_split[, 1], 1, 1)

# Combine the first letter ". " and last name
str_c(abb_first, ". ", names_split[, 2])
```

```
## [1] "D. Prince" "C. Kent"
```

2.12 Final challenges

Again, we deepen the learned knowledge in this exercise.

```r
# Define some full names
names <- c("Diana Prince", "Clark Kent")

# Split into first and last names
names_split <- str_split(names, fixed(" "), simplify = TRUE)

# Extract the first letter in the first name
abb_first <- str_sub(names_split[, 1], 1, 1)
```

```r
# Combine the first letter ". " and last name
str_c(abb_first, ". ", names_split[, 2])
```

```
## [1] "D. Prince" "C. Kent"
```

```r
# Use all names in babynames_2014
all_names <- babynames_2014$name

# Get the last two letters of all_names
last_two_letters <- str_sub(all_names, -2, -1)

# Does the name end in "ee"?
ends_in_ee <- str_detect(last_two_letters, "ee")

# Extract rows and "sex" column
sex <- str_sub(babynames_2014$sex[ends_in_ee])

# Display result as a table
table(sex)
```

```
## sex
##   F   M
## 572  84
```

3 Pattern matching with regular expression

3.1 Matching the start or end of the string

In this exercise we learn how to concatenate regex patterns with %R% and also that the constants START and END are predefined as the symbols that regex uses for start and end line detection.

```r
library(rebus)
```

```
##
## Attaching package: 'rebus'

## The following object is masked from 'package:stringr':
##
##     regex
```

```r
library(htmlwidgets)
# Some strings to practice with
x <- c("cat", "coat", "scotland", "tic toc")

# Print END
END
```

```
## <regex> $
```

```r
# Run me
str_view(x, pattern = START %R% "c")
```

```r
# Match the strings that start with "co"
str_view(x, pattern = START %R% "co")
```

```r
# Match the strings that end with "at"
str_view(x, pattern =  "at" %R% END)
```

```r
# Match the string that is exactly "cat"
str_view(x, pattern = START %R% "cat" %R% END)
```

3.2 Matching any character

ANY_CHAR is the constant that contains the regex symbol for any character (a.k.a. the wildcard).

```r
# Match two characters, where the second is a "t"
str_view(x, pattern = ANY_CHAR %R% "t")
```

```r
# Match a "t" followed by any character
str_view(x, pattern = "t" %R% ANY_CHAR)
```

```r
# Match two characters
str_view(x, pattern = ANY_CHAR %R% ANY_CHAR)
```

```r
# Match a string with exactly three characters
str_view(x, pattern = START %R% ANY_CHAR %R% ANY_CHAR %R% ANY_CHAR %R% END)
```

3.3 Combining with stringr functions

In this exercise we combine stringr functions with regex patterns.

```r
pattern <- "q" %R% ANY_CHAR

# Find names that have the pattern
names_with_q <- str_subset(boy_names, pattern)

# How many names were there?
length(names_with_q)
```

```
## [1] 96
```

```r
# Find part of name that matches pattern
part_with_q <- str_extract(boy_names, pattern)

# Get a table of counts
table(part_with_q)
```

```
## part_with_q
## qa qe qi qm qo qu
##  1  1  2  2  1 89
```

```r
# Did any names have the pattern more than once?
count_of_q <- str_count(boy_names, pattern)

# Get a table of counts
table(count_of_q)
```

```
## count_of_q
##     0     1
## 13951    96
```

```r
# Which babies got these names?
with_q <- str_detect(boy_names, pattern)

# What fraction of babies got these names?
mean(with_q)
```

```
## [1] 0.006834199
```

3.4 Alternation

or() can be used to specify alternative patterns (or parts of a pattern).

```
# Match Jeffrey or Geoffrey
whole_names <- or("Jeffrey", "Geoffrey")
str_view(boy_names, pattern = whole_names, match = TRUE)
```

```
# Match Jeffrey or Geoffrey, another way
common_ending <- or("Je", "Geo") %R% "ffrey"
str_view(boy_names, pattern = common_ending, match = TRUE)
```

```
# Match with alternate endings
by_parts <- or("Je", "Geo") %R% "ff" %R% or("ry", "ery", "rey", "erey")
str_view(boy_names, pattern = by_parts, match = TRUE)
```

```
# Match names that start with Cath or Kath
ckath <- or("Cath", "Kath")
str_view(girl_names, pattern = ckath, match = TRUE)
```

3.5 Character classes

char_class() is a way to specify that that any character that is contained in the specified string should match.

```
# Create character class containing vowels
vowels <- char_class("aeiouAEIOU")

# Print vowels
vowels
```

```
## <regex> [aeiouAEIOU]
```

```
# See vowels in x with str_view()
str_view(x, vowels)
```

```
# See vowels in x with str_view_all()
str_view_all(x, vowels)
```

```
# Number of vowels in boy_names
num_vowels <- str_count(boy_names, vowels)

# Number of characters in boy_names
name_length <- str_length(boy_names)
# Calc mean number of vowels
mean(str_count(boy_names, vowels))
```

```
## [1] 2.385563
```

```
# Calc mean fraction of vowels per name
mean(num_vowels / name_length)
```

```
## [1] 0.4000596
```

3.6 Repetition

In this exercise functions for one or more occurrences and zero or more occurrences of a pattern are introduced.

```r
# Vowels from last exercise
vowels <- char_class("aeiouAEIOU")

# See names with only vowels
str_view(boy_names,
  pattern = exactly(one_or_more(vowels)),
  match = TRUE)
```

```r
# Use `negated_char_class()` for everything but vowels
not_vowels <- negated_char_class("aeiouAEIOU")

# See names with no vowels
str_view(boy_names,
  pattern = exactly(one_or_more(not_vowels)),
  match = TRUE)
```

3.7 Hunting for phone numbers

In this exercise we create a complex pattern to match phone numbers and use str_extract and str_extract all to show all matches.

```r
# Create a separator pattern
separator <- char_class("-.() ")

# Test it
str_view_all(contact, pattern = separator)
# Create a separator pattern
separator <- char_class("-.() ")

# Test it
str_view_all(contact, pattern = separator)
# Use these components
three_digits <- DGT %R% DGT %R% DGT
four_digits <- three_digits %R% DGT
separator <- char_class("-.() ")

# Create phone pattern
phone_pattern <- zero_or_more(OPEN_PAREN) %R%
  three_digits %R%
  zero_or_more(separator) %R%
  three_digits %R%
  zero_or_more(separator) %R%
  four_digits

# Test it
str_view_all(contact, pattern = phone_pattern)
# Use this pattern
three_digits <- DGT %R% DGT %R% DGT
four_digits <- three_digits %R% DGT
separator <- char_class("-.() ")
phone_pattern <- optional(OPEN_PAREN) %R%
  three_digits %R%
  zero_or_more(separator) %R%
  three_digits %R%
  zero_or_more(separator) %R%
  four_digits

# Extract phone numbers
str_extract(contact, phone_pattern)

# Extract ALL phone numbers
str_extract_all(contact, phone_pattern)
```

3.8 Extracting age and gender from accident narratives

We practice our previously acquired knowledge of regex.

```
# Pattern to match one or two digits
age <- or(ANY_CHAR %R% ANY_CHAR, ANY_CHAR)
narratives = readRDS("narratives.rds")

# Test it
str_view(narratives, pattern = age)
```

```
# Use this pattern
age <- DGT %R% optional(DGT)

# Pattern to match units
unit <- zero_or_more(" ") %R% or("YO", "YR", "MO")

# Test pattern with age then units
str_view(narratives, pattern = age %R% unit)
```

```
# Use these patterns
age <- DGT %R% optional(DGT)
unit <- optional(SPC) %R% or("YO", "YR", "MO")

# Pattern to match gender
gender <- zero_or_more(" ") %R% or("M", "F")

# Test pattern with age then units then gender
str_view(narratives, pattern = age %R% unit %R% gender)
```

```
# Use these patterns
age <- DGT %R% optional(DGT)
unit <- optional(SPC) %R% or("YO", "YR", "MO")
gender <- optional(SPC) %R% or("M", "F")

# Extract age, unit, gender
str_extract(narratives, age %R% unit %R% gender)
```

```
##  [1] "19YOM"   "31 YOF"  "82 YOM"  "33 YOF"  "10YOM"   "53 YO F" "13 MOF"
##  [8] "14YR M"  "55YOM"   "5 YOM"
```

3.9 Parsing age and gender into pieces

In this exercise we detect units and convert the data accordingly to years.

```
# age_gender, age, gender, unit are pre-defined
ls.str()
```

```r
# Extract age and make numeric
as.numeric(str_extract(age_gender, age))
# Replace age and units with ""
genders <- str_remove(age_gender, pattern = age %R% unit)

# Replace extra spaces
str_remove_all(genders, pattern = one_or_more(SPC))
# Numeric ages, from previous step
ages_numeric <- as.numeric(str_extract(age_gender, age))

# Extract units
time_units <- str_extract(age_gender, unit)

# Extract first word character
time_units_clean <- str_extract(time_units, WRD)

# Turn ages in months to years
ifelse(time_units_clean == "Y", ages_numeric, ages_numeric/12)
```

4 More advanced matching and manipulation

4.1 Capturing parts of a pattern

We learn how to use the capture function in R and also introduce str_match to pull out matches.

```r
# Capture parts between @ and . and after .
email <- capture(one_or_more(WRD)) %R%
  "@" %R% capture(one_or_more(WRD)) %R%
  DOT %R% capture(one_or_more(WRD))

# Check match hasn't changed
str_view(hero_contacts, email)
# Pattern from previous step
email <- capture(one_or_more(WRD)) %R%
  "@" %R% capture(one_or_more(WRD)) %R%
  DOT %R% capture(one_or_more(WRD))

# Pull out match and captures
email_parts <- str_match(hero_contacts, email)
email_parts

# Save host
host <- email_parts[,3]
host
```

4.2 Pulling out parts of a phone number

In this exercise we use capture and str_match to pull out parts of phone numbers and then add them back together to a specified format.

```r
# View text containing phone numbers
contact

# Add capture() to get digit parts
```

```
phone_pattern <- capture(three_digits) %R% zero_or_more(separator) %R%
           capture(three_digits) %R% zero_or_more(separator) %R%
           capture(four_digits)

# Pull out the parts with str_match()
phone_numbers <- str_match(contact, phone_pattern)

# Put them back together
str_c(
  "(",
  phone_numbers[,2],
  ") ",
  phone_numbers[,3],
  "-",
  phone_numbers[,4])
```

4.3 Extracting age and gender again

We're revisiting capture and str_match.

```
# narratives has been pre-defined
narratives
```

```
##  [1] "19YOM-SHOULDER STRAIN-WAS TACKLED WHILE PLAYING FOOTBALL W/ FRIENDS "
##  [2] "31 YOF FELL FROM TOILET HITITNG HEAD SUSTAINING A CHI "
##  [3] "ANKLE STR. 82 YOM STRAINED ANKLE GETTING OUT OF BED "
##  [4] "TRIPPED OVER CAT AND LANDED ON HARDWOOD FLOOR. LACERATION ELBOW, LEFT. 33 YOF*"
##  [5] "10YOM CUT THUMB ON METAL TRASH CAN DX AVULSION OF SKIN OF THUMB "
##  [6] "53 YO F TRIPPED ON CARPET AT HOME. DX HIP CONTUSION "
##  [7] "13 MOF TRYING TO STAND UP HOLDING ONTO BED FELL AND HIT FOREHEAD ON RADIATOR DX LACERATION"
##  [8] "14YR M PLAYING FOOTBALL; DX KNEE SPRAIN "
##  [9] "55YOM RIDER OF A BICYCLE AND FELL OFF SUSTAINED A CONTUSION TO KNEE "
## [10] "5 YOM ROLLING ON FLOOR DOING A SOMERSAULT AND SUSTAINED A CERVICAL STRA IN"
```

```
# Add capture() to get age, unit and sex
pattern <- capture(optional(DGT) %R% DGT) %R%
  optional(SPC) %R% capture(or("YO", "YR", "MO")) %R%
  optional(SPC) %R% capture(or("M", "F"))

# Pull out from narratives
str_match(narratives, pattern)
```

```
##       [,1]      [,2] [,3] [,4]
##  [1,] "19YOM"   "19" "YO" "M"
##  [2,] "31 YOF"  "31" "YO" "F"
##  [3,] "82 YOM"  "82" "YO" "M"
##  [4,] "33 YOF"  "33" "YO" "F"
##  [5,] "10YOM"   "10" "YO" "M"
##  [6,] "53 YO F" "53" "YO" "F"
##  [7,] "13 MOF"  "13" "MO" "F"
##  [8,] "14YR M"  "14" "YR" "M"
##  [9,] "55YOM"   "55" "YO" "M"
## [10,] "5 YOM"   "5"  "YO" "M"
```

```r
# Edit to capture just Y and M in units
pattern2 <- capture(optional(DGT) %R% DGT) %R%
  optional(SPC) %R% capture(or("Y", "M")) %R% optional(or("O","R")) %R%
  optional(SPC) %R% capture(or("M", "F"))

# Check pattern
str_view(narratives, pattern2)
```

```r
# Pull out pieces
str_match(narratives, pattern2)
```

```
##       [,1]      [,2] [,3] [,4]
##  [1,] "19YOM"   "19" "Y"  "M"
##  [2,] "31 YOF"  "31" "Y"  "F"
##  [3,] "82 YOM"  "82" "Y"  "M"
##  [4,] "33 YOF"  "33" "Y"  "F"
##  [5,] "10YOM"   "10" "Y"  "M"
##  [6,] "53 YO F" "53" "Y"  "F"
##  [7,] "13 MOF"  "13" "M"  "F"
##  [8,] "14YR M"  "14" "Y"  "M"
##  [9,] "55YOM"   "55" "Y"  "M"
## [10,] "5 YOM"   "5"  "Y"  "M"
```

4.4 Using backreferences in patterns

In this exercise backreferences (REF1-REF9) are introduced, which can be used in combination with capture to make sure that repeated occurrences can be matched.

```r
# Names with three repeated letters
repeated_three_times <- capture(LOWER) %R% REF1 %R% REF1

# Test it
str_view(boy_names, pattern = repeated_three_times, match = TRUE)
```

```r
# Names with a pair of repeated letters
pair_of_repeated <- capture(LOWER %R% LOWER) %R% REF1

# Test it
str_view(boy_names, pattern = pair_of_repeated, match = TRUE)
```

```r
# Names with a pair that reverses
pair_that_reverses <- capture(LOWER) %R% capture(LOWER) %R% REF2 %R% REF1

# Test it
str_view(boy_names, pattern = pair_that_reverses, match = TRUE)
```

```r
# Four letter palindrome names
four_letter_palindrome <- exactly(capture(LOWER) %R% capture(LOWER) %R% REF2 %R% REF1)

# Test it
str_view(boy_names, pattern = four_letter_palindrome, match = TRUE)
```

4.5 Replacing with regular expressions

Using str_replace we can replace matched patterns either with the same string or one of a whole vector of strings.

```r
# View text containing phone numbers
contact

# Replace digits with "X"
str_replace(contact, DGT, "X")

# Replace all digits with "X"
str_replace_all(contact, DGT, "X")

# Replace all digits with different symbol
str_replace_all(contact, DGT, c("X", ".", "*", "_"))
```

4.6 Replacing with backreferences

In the code below we capture words and then replace them with “carelessly” + themselves using references.

```r
adverbs = readRDS("adverbs.rds")
# Build pattern to match words ending in "ING"
pattern <- one_or_more(WRD) %R% "ING"
str_view(narratives, pattern)
```

```r
# Test replacement
str_replace(narratives, capture(pattern),
  str_c("CARELESSLY", REF1, sep = " "))
```

```
##  [1] "19YOM-SHOULDER STRAIN-WAS TACKLED WHILE CARELESSLY PLAYING FOOTBALL W/ FRIENDS "
##  [2] "31 YOF FELL FROM TOILET HITITNG HEAD CARELESSLY SUSTAINING A CHI "
##  [3] "ANKLE STR. 82 YOM STRAINED ANKLE CARELESSLY GETTING OUT OF BED "
##  [4] "TRIPPED OVER CAT AND LANDED ON HARDWOOD FLOOR. LACERATION ELBOW, LEFT. 33 YOF*"
##  [5] "10YOM CUT THUMB ON METAL TRASH CAN DX AVULSION OF SKIN OF THUMB "
##  [6] "53 YO F TRIPPED ON CARPET AT HOME. DX HIP CONTUSION "
##  [7] "13 MOF CARELESSLY TRYING TO STAND UP HOLDING ONTO BED FELL AND HIT FOREHEAD ON RADIATOR DX LAC
##  [8] "14YR M CARELESSLY PLAYING FOOTBALL; DX KNEE SPRAIN "
##  [9] "55YOM RIDER OF A BICYCLE AND FELL OFF SUSTAINED A CONTUSION TO KNEE "
## [10] "5 YOM CARELESSLY ROLLING ON FLOOR DOING A SOMERSAULT AND SUSTAINED A CERVICAL STRA IN"
```

```r
# One adverb per narrative
adverbs_10 <- sample(adverbs, 10)

# Replace "***ing" with "adverb ***ly"
str_replace(narratives,
  capture(pattern),
  str_c(adverbs_10, REF1, sep = " "))
```

```
##  [1] "19YOM-SHOULDER STRAIN-WAS TACKLED WHILE COURAGEOUSLY PLAYING FOOTBALL W/ FRIENDS "
##  [2] "31 YOF FELL FROM TOILET HITITNG HEAD MOSTLY SUSTAINING A CHI "
##  [3] "ANKLE STR. 82 YOM STRAINED ANKLE LIVELY GETTING OUT OF BED "
##  [4] "TRIPPED OVER CAT AND LANDED ON HARDWOOD FLOOR. LACERATION ELBOW, LEFT. 33 YOF*"
##  [5] "10YOM CUT THUMB ON METAL TRASH CAN DX AVULSION OF SKIN OF THUMB "
##  [6] "53 YO F TRIPPED ON CARPET AT HOME. DX HIP CONTUSION "
##  [7] "13 MOF JUBILANTLY TRYING TO STAND UP HOLDING ONTO BED FELL AND HIT FOREHEAD ON RADIATOR DX LAC
##  [8] "14YR M FRANKLY PLAYING FOOTBALL; DX KNEE SPRAIN "
##  [9] "55YOM RIDER OF A BICYCLE AND FELL OFF SUSTAINED A CONTUSION TO KNEE "
## [10] "5 YOM WARMLY ROLLING ON FLOOR DOING A SOMERSAULT AND SUSTAINED A CERVICAL STRA IN"
```

4.7 Matching a specific code point or code groups

Here stri_trans_nfd is introduced, which decomposes letters with accents into separate letter and accent characters.

```r
library(stringi)
# Names with builtin accents
(tay_son_builtin <- c(
  "Nguy\u1ec5n Nh\u1ea1c",
  "Nguy\u1ec5n Hu\u1ec7",
  "Nguy\u1ec5n Quang To\u1ea3n"
))

# Convert to separate accents
tay_son_separate <- stri_trans_nfd(tay_son_builtin)

# Verify that the string prints the same
tay_son_separate

# Match all accents
str_view_all(tay_son_separate, UP_DIACRITIC)
```

4.8 Matching a single grapheme

In this exercise we introduce the opposite of stri_trans_nfd, which is stri_trans_nfc.

```r
# tay_son_separate has been pre-defined
tay_son_separate

# View all the characters in tay_son_separate
str_view_all(tay_son_separate, ANY_CHAR)
# View all the graphemes in tay_son_separate
str_view_all(tay_son_separate, GRAPHEME)
# Combine the diacritics with their letters
tay_son_builtin <- stri_trans_nfc(tay_son_separate)
tay_son_builtin

# View all the graphemes in tay_son_builtin
str_view_all(tay_son_builtin, GRAPHEME)
```

5 Case studies

5.1 Getting the play into R

In this exercise we read in a play and section it with the functions that we've learned in the previous chapters.

```r
# Read play in using stri_read_lines()
library(stringi)
library(stringr)
earnest_file = "importance-of-being-earnest.txt"
earnest <- stri_read_lines(earnest_file)

# Detect start and end lines
start <- str_which(earnest, fixed("START OF THE PROJECT"))
end <- str_which(earnest, fixed("END OF THE PROJECT"))
```

```r
# Get rid of gutenberg intro text
earnest_sub  <- earnest[(start + 1):(end - 1)]

# Detect first act
lines_start <- str_which(earnest_sub, fixed("FIRST ACT"))

# Set up index
intro_line_index <- 1:(lines_start - 1)

# Split play into intro and play
intro_text <- earnest_sub[intro_line_index]
play_text <- earnest_sub[-intro_line_index]

# Take a look at the first 20 lines
writeLines(play_text[1:20])
```

```
## FIRST ACT
##
##
## SCENE
##
##
## Morning-room in Algernon's flat in Half-Moon Street.  The room is
## luxuriously and artistically furnished.  The sound of a piano is heard in
## the adjoining room.
##
## [Lane is arranging afternoon tea on the table, and after the music has
## ceased, Algernon enters.]
##
## Algernon.  Did you hear what I was playing, Lane?
##
## Lane.  I didn't think it polite to listen, sir.
##
## Algernon.  I'm sorry for that, for your sake.  I don't play
## accurately--any one can play accurately--but I play with wonderful
## expression.  As far as the piano is concerned, sentiment is my forte.  I
```

5.2 Identifying the lines, take 1

Again, we practice the previously acquired knowledge. Furthermore we learn that ascii_upper() matches to a capital character.

```r
# Pattern for start, word then .
pattern_1 <- START %R% one_or_more(WRD) %R% DOT

# Test pattern_1
str_view(play_lines, pattern_1, match = T)
str_view(play_lines, pattern_1, match = F)

# Pattern for start, capital, word then .
pattern_2 <- START %R% ascii_upper() %R% one_or_more(WRD) %R% DOT

# Test pattern_2
str_view(play_lines, pattern_2, match = T)
str_view(play_lines, pattern_2, match = F)
# Pattern from last step
pattern_2 <- START %R% ascii_upper() %R% one_or_more(WRD) %R% DOT

# Get subset of lines that match
lines <- str_subset(play_lines, pattern_2)

# Extract match from lines
who <- str_extract(lines, pattern_2)

# Let's see what we have
unique(who)
```

5.3 Identifying the lines, take 2

Again, we practice the previously acquired knowledge. Furthermore we learn that or1 similar to or, except it matches only when just one of the strings match.

```r
# Variables from previous step
characters <- c("Algernon", "Jack", "Lane", "Cecily", "Gwendolen", "Chasuble",
  "Merriman", "Lady Bracknell", "Miss Prism")
pattern_3 <- START %R% or1(characters) %R% DOT

# Pull out matches
lines <- str_subset(play_lines, pattern_3)

# Extract match from lines
who <- str_extract(lines, pattern_3)

# Let's see what we have
unique(who)

# Count lines per character
table(who)
```

5.4 Changing case to ease matching

We learn how we can change lower to/from upper case to get the correct match.

```r
catcidents = readRDS("catcidents.rds")
# catcidents has been pre-defined
head(catcidents)
```

```
## [1] "79yOf Fractured fingeR tRiPPED ovER cAT ANd fell to FlOOr lAst nIGHT AT HOME*"
## [2] "21 YOF REPORTS SUS LACERATION OF HER LEFT HAND WHEN SHE WAS OPENING A CAN OF CAT FOOD JUST PTA.
## [3] "87YOF TRIPPED OVER CAT, HIT LEG ON STEP. DX LOWER LEG CONTUSION "
## [4] "bLUNT CHest trAUma, R/o RIb fX, R/O CartiLAgE InJ To RIB cAge; 32YOM walKiNG DOG, dog took OfF
## [5] "42YOF TO ER FOR BACK PAIN AFTER PUTTING DOWN SOME CAT LITTER DX: BACK PAIN, SCIATICA"
## [6] "4YOf DOg jUst hAd PUpPieS, Cat TRIED 2 get PuPpIes, pT THru CaT dwn stA Irs, LoST foOTING & FEL
```

```r
# Construct pattern of DOG in boundaries
whole_dog_pattern <- whole_word("DOG")

# See matches to word DOG
str_view(catcidents, whole_dog_pattern, match=T)
```

```
# From previous step
whole_dog_pattern <- whole_word("DOG")

# Transform catcidents to upper case
catcidents_upper <- str_to_upper(catcidents)

# View matches to word "DOG" again
str_view(catcidents_upper, whole_dog_pattern, match=T)
```

```
# From previous steps
whole_dog_pattern <- whole_word("DOG")
catcidents_upper <- str_to_upper(catcidents)

# Which strings match?
has_dog <- str_detect(catcidents_upper, whole_dog_pattern)

# Pull out matching strings in original
catcidents[has_dog]
```

```
##  [1] "bLUNT CHest trAUma, R/o RIb fX, R/O CartiLAgE InJ To RIB cAge; 32YOM walKiNG DOG, dog took OfF
##  [2] "4YOf DOg jUst hAd PUpPieS, Cat TRIED 2 get PuPpIes, pT THru CaT dwn stA Irs, LoST foOTING & FE
##  [3] "unhelmeted 14yof riding her bike with her dog when she saw a cat and sw erved c/o head/shoulde
##  [4] "Rt Shoulder Strain.26Yof Was Walking Dog On Leash And Dot Saw A Cat And Pulled Leash."
##  [5] "67 YO F WENT TO WALK DOG, IT STARTED TO CHASE CAT JERKED LEASH PULLED H ER OFF PATIO, FELL HUR
##  [6] "46yof taking dog outside, dog bent her fingers back on a door. dog jerk ed when saw cat. hand 
##  [7] "PUSHING HER UTD WITH SHOTS DOG AWAY FROM THE CAT'S BOWL&BITTEN TO FINGE R>>PW/DOG BITE"
##  [8] "DX R SH PN: 27YOF W/ R SH PN X 5D. STATES WAS YANK' BY HER DOG ON LEASH W DOG RAN AFTER CAT; W
##  [9] "39Yof dog pulled her down the stairs while chasing a cat dx: rt ankle inj"
## [10] "44Yof Walking Dog And The Dof Took Off After A Cat And Pulled Pt Down B Y The Leash Strained N
```

5.5 Ignoring case when matching

With regex and the param ignore_case it's also possible to match in a case insensitive way.

```
# View matches to "TRIP"
str_view(catcidents, "TRIP", match=T)
```

```
# Construct case insensitive pattern
trip_pattern <- regex("TRIP", ignore_case = TRUE)

# View case insensitive matches to "TRIP"
str_view(catcidents, trip_pattern, match=T)
```

```
# From previous step
trip_pattern <- regex("TRIP", ignore_case = TRUE)

# Get subset of matches
trip <- str_subset(catcidents, trip_pattern)

# Extract matches
str_extract(trip, trip_pattern)
```

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

5.6 Fixing case problems

It's also possible to transform strings to a common case, but not upper or lower case. Here we use that to transform to title case and sentence case.

```
library(stringi)

# Get first five catcidents
cat5 <- catcidents[1:5]

# Take a look at original
writeLines(cat5)
```

```
## 79yOf Fractured fingeR tRiPPED ovER cAT ANd fell to FlOOr lAst nIGHT AT HOME*
## 21 YOF REPORTS SUS LACERATION OF HER LEFT HAND WHEN SHE WAS OPENING A CAN OF CAT FOOD JUST PTA. DX H
## 87YOF TRIPPED OVER CAT, HIT LEG ON STEP. DX LOWER LEG CONTUSION
## bLUNT CHest trAUma, R/o RIb fX, R/O CartiLAgE InJ To RIB cAge; 32YOM walKiNG DOG, dog took OfF aFtER
## 42YOF TO ER FOR BACK PAIN AFTER PUTTING DOWN SOME CAT LITTER DX: BACK PAIN, SCIATICA
```

```
# Transform to title case
writeLines(str_to_title(cat5))
```

```
## 79yof Fractured Finger Tripped Over Cat And Fell To Floor Last Night At Home*
## 21 Yof Reports Sus Laceration Of Her Left Hand When She Was Opening A Can Of Cat Food Just Pta. Dx H
## 87yof Tripped Over Cat, Hit Leg On Step. Dx Lower Leg Contusion
## Blunt Chest Trauma, R/O Rib Fx, R/O Cartilage Inj To Rib Cage; 32yom Walking Dog, Dog Took Off After
## 42yof To Er For Back Pain After Putting Down Some Cat Litter Dx: Back Pain, Sciatica
```

```
# Transform to title case with stringi
stri_trans_totitle(cat5)
```

```
## [1] "79yof Fractured Finger Tripped Over Cat And Fell To Floor Last Night At Home*"
## [2] "21 Yof Reports Sus Laceration Of Her Left Hand When She Was Opening A Can Of Cat Food Just Pta.
## [3] "87yof Tripped Over Cat, Hit Leg On Step. Dx Lower Leg Contusion "
## [4] "Blunt Chest Trauma, R/O Rib Fx, R/O Cartilage Inj To Rib Cage; 32yom Walking Dog, Dog Took Off
## [5] "42yof To Er For Back Pain After Putting Down Some Cat Litter Dx: Back Pain, Sciatica"
```

```
# Transform to sentence case with stringi
writeLines(stri_trans_totitle(cat5, type = "sentence"))
```

```
## 79yof fractured finger tripped over cat and fell to floor last night at home*
## 21 yof reports sus laceration of her left hand when she was opening a can of cat food just pta. Dx h
## 87yof tripped over cat, hit leg on step. Dx lower leg contusion
## Blunt chest trauma, r/o rib fx, r/o cartilage inj to rib cage; 32yom walking dog, dog took off after
## 42yof to er for back pain after putting down some cat litter dx: back pain, sciatica
```