**STRING MANIPULATION IM R WITH STRINGR**

Guidelines to write strings: simple "", ' " ' to write double quotes. "\"\"" if ' is in the text.

# 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 one end of the table."

# Putting lines in a vector

lines <- c(line1, line2, line3)

# Print lines

lines

print(lines)

# Use writeLines() on lines --> writes what is within the first ""

writeLines(lines)

# Write lines with a space separator --Y writes the lines not on newline.

writeLines(lines, sep = "")

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

writeLines("hello\n\U1F30D")

cat() is similar to writeLines() but separates elements with a space by space and tries to convert non-character objects to strings.

In "hello\n\U1F30D" there are two escape sequences: \n gives a newline, and \Ufollowed by up to 8 hex digits sequence denotes a particular Unicode character. \ is called an escape sequence

# Should display: To have a \ you need \\

writeLines("To have a \\ you need \\\\")

# Should display:

# This is a really

# really really

# long string

writeLines("This is a really \n really really\n long string")

# Hello world in Hindi

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

?Quotes

Using format with numbers to later display them in strings:

# 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)

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

format(c(1.0011, 2.011, 1), digits = 1)

# Format percent\_change to one place after the decimal point

format(percent\_change, digits = 2)

# Format income to whole numbers

format(income, digits = 2)

# Format p\_values in fixed format

format(p\_values, scientific = FALSE)

formatted\_income <- format(income, digits = 2)

print(formatted\_income)

writeLines(formatted\_income)

trimmed\_income <- format(income, digits = 2, trim= TRUE)

writeLines(trimmed\_income)

pretty\_income <- format(income, digits = 2, big.mark = ",", big.interval = 3, scientific = FALSE)

writeLines(pretty\_income)

formatC() – an alternative way of formating numbers based on 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, format argument: "f" for fixed,"e" for scientific,and "g" for fixed unless scientific saves space

formatC(x, format = "f", digits = 1)

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

formatC(y, format = "f", digits = 1)

# Format percent\_change to one place after the decimal point

formatC(percent\_change, format = "f", digits = 1)

# percent\_change with flag = "+", display of the sign (flag = "+"), left align numbers (flag = "-") and pad numbers with leading zeros (flag = "0")

formatC(percent\_change, format = "f", digits = 1, flag = "+")

# Format p\_values using format = "g" and digits = 2

formatC(p\_values, format = "g", digits = 2)

paste() takes strings as imput ant prints them togheter

# 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 <- c("2010: +4.0%", "2011: -1.9%", "2012: +3.0%", "2013: -5.0%")

# Collapse all years into single string

paste(year\_percent, collapse = ", ")

Create e table based on strings

# 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 ="")

# Create formatted\_names

formatted\_names <- format(income\_names, justify = "right")

# Create rows

rows <- paste(formatted\_names, dollar\_income, sep = " ")

# Write rows

writeLines(rows)

Order a pizza with random toppings:

# Randomly sample 3 toppings

my\_toppings <- sample(toppings, size = 3)

my\_toppings <- c("anchovies", "garlic", "pepperoncini")

print(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)

STRING R

library(stringr)

str\_c() – paste function as before, but with no separator as default

str\_lenght()

str\_sub()

library(stringr)

my\_toppings <- c("cheese", NA, NA)

my\_toppings\_and <- paste(c("", "", "and "), my\_toppings, sep = "")

my\_toppings\_and

# Use str\_c() instead of paste(): my\_toppings\_str

my\_toppings\_str <- str\_c(c("", "", "and "), my\_toppings)

my\_toppings\_str

# paste() my\_toppings\_and with collapse = ", "

paste(my\_toppings\_and, collapse = ", ")

# str\_c() my\_toppings\_str with collapse = ", "

str\_c(my\_toppings\_str, collapse = ", ")

str\_replace\_na() relace missing values with a string of your choice.

str\_length() gives the length (in characters) of each string in the input.

library(stringr)

library(babynames)

library(dplyr)

# 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)

# Find the length of all boy\_names

boy\_length <- str\_length(boy\_names)

head(boy\_length)

# 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)

# Confirm str\_length() works with factors

head(str\_length(factor(boy\_names)))

**[str\_sub()](https://www.rdocumentation.org/packages/stringr/topics/str_sub" \t "_blank)** function in stringr extracts parts of strings based on their location. [**str\_sub()**](https://www.rdocumentation.org/packages/stringr/topics/str_sub) function in stringr extracts parts of strings based on their location. the first argument, string, is a vector of strings. The arguments start and end specify the boundaries of the piece to extract in characters.

str\_sub(c("Bruce", "Wayne"), 1, 4)

str\_sub(c("Bruce", "Wayne"), -4, -1)

string functions that look for matches and take a pattern argument:

str\_detect() Returns TRUE or FALSE based wether or not the pattern argument occurs in the strings

str\_subset() return only the strings that contain the vector

str\_count() number of times a pattern in each string. (Vector of the same length as imput vector returning the numbers of e.g. how many times a letter occurs in each string!).

# Look for pattern "zz" in boy\_names

contains\_zz <- str\_detect(boy\_names, pattern = fixed("zz"))

# How many names contain "zz"?

sum(contains\_zz) #sum counts TRUE as 1 and FALSE as 0

# Which names contain "zz"?

str\_subset(boy\_names, pattern = fixed("zz"))

# Which rows in boy\_df have names that contain "zz"?

boy\_df[contains\_zz,]

# Find girl\_names that contain "U", pettern is case specific so capital letters only occur at the start

starts\_U <- str\_subset(girl\_names, pattern = fixed("U"))

# Find girl\_names that contain "U" and "z" 🡪 just feed the previous substring into next filter

str\_subset(starts\_U, pattern = fixed("z"))

# Count occurrences of "a" in girl\_names

number\_as <- str\_count(girl\_names, pattern = fixed("a"))

# Count occurrences of "A" in girl\_names

number\_As <- str\_count(girl\_names, pattern = fixed("A"))

# Find total "a" + "A"

total\_as <- number\_as + number\_As

# girl\_names with more than 4 a's

subset(girl\_names, total\_as>4)

str\_split() – Splits a string based on a pattern (usually returns a list, unless simplify = True, then a matrix is returned.

e.g.

date\_ranges <- c("23.01.2017 - 29.01.2017", "30.01.2017 - 06.02.2017")

split\_dates <- str\_split(date\_ranges, pattern=fixed("-"))

split\_dates\_n <- str\_split(date\_ranges, pattern=fixed(" - "), simplify = TRUE, n = 2)

# 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, pattern=fixed("."), simplify=TRUE)

or

# 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)