Lab Assignment: Chapter 16 - Manipulating Strings

Michael Beebe

library(glue)  
library(stringr)  
library(purrr)

## 1. paste

Using the paste function, paste together “Hello” and “World”

* Use no separator
* Use a dash as a separator

hello <- "Hello"  
world <- "World"  
  
paste(hello, world, sep = "-")

## [1] "Hello-World"

String concatenation is somewhat outdated. I find string interpolation to be easier and much more readable. We can use the “glue” library to do this:

name <- "Michael"  
glue("My name is {name}.")

## My name is Michael.

## 2. String Manipulation

Research the following string manipulation functions:

nchar() Takes a character vector as an argument and returns a vector whose elements contain the sizes of the corresponding elements of x.

nchar("Hello World")

## [1] 11

tolower() Translate characters in character vectors, in particular from upper to lower case or vice versa.

tolower("HELLO WORLD")

## [1] "hello world"

toupper() Translate characters in character vectors, in particular from upper to lower case or vice versa.

toupper("hello world")

## [1] "HELLO WORLD"

chartr() Translate characters in character vectors, in particular from upper to lower case or vice versa.

x <- "MiXeD cAsE 123"  
chartr("iXs", "why", x)

## [1] "MwheD cAyE 123"

chartr("a-cX", "D-Fw", x)

## [1] "MiweD FAsE 123"

str\_replace() replaces the values in x with indices given in list by those given in values. If necessary, the values in values are recycled.

fruits <- c("one apple", "two pears", "three bananas")  
str\_replace(fruits, "([aeiou])", "")

## [1] "ne apple" "tw pears" "thre bananas"

substr() Extract or replace substrings in a character vector.

substr("abcdef", 2, 4)

## [1] "bcd"

str\_sub() will recycle all arguments to be the same length as the longest argument. If any arguments are of length 0, the output will be a zero length character vector.

hw <- "Hadley Wickham"  
str\_sub(hw, 1, 6)

## [1] "Hadley"

setdiff() Each of union, intersect, setdiff and setequal will discard any duplicated values in the arguments, and they apply as.vector to their arguments (and so in particular coerce factors to character vectors).

(x <- c(sort(sample(1:20, 9)), NA))

## [1] 1 3 4 13 14 15 17 18 20 NA

(y <- c(sort(sample(3:23, 7)), NA))

## [1] 4 8 9 11 12 13 23 NA

setdiff(x, y)

## [1] 1 3 14 15 17 18 20

setequal() Performs set union, intersection, (asymmetric!) difference, equality and membership on two vectors.

(x <- c(sort(sample(1:20, 9)), NA))

## [1] 2 4 6 7 9 10 14 16 19 NA

(y <- c(sort(sample(3:23, 7)), NA))

## [1] 15 16 18 19 20 22 23 NA

setequal(x, y)

## [1] FALSE

abbreviate() Abbreviate strings to at least minlength characters, such that they remain unique (if they were), unless strict = TRUE.

x <- c("abcd", "efgh", "abce")  
abbreviate(x, 2)

## abcd efgh abce   
## "abcd" "ef" "abce"

strsplit() Split the elements of a character vector x into substrings according to the matches to substring split within them.

x <- c(as = "asfef", qu = "qwerty", "yuiop[", "b", "stuff.blah.yech")  
strsplit(x, "e")

## $as  
## [1] "asf" "f"   
##   
## $qu  
## [1] "qw" "rty"  
##   
## [[3]]  
## [1] "yuiop["  
##   
## [[4]]  
## [1] "b"  
##   
## [[5]]  
## [1] "stuff.blah.y" "ch"

sub() grep, grepl, regexpr, gregexpr and regexec search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

str <- "Now is the time "  
sub(" +$", "", str)

## [1] "Now is the time"

gsub() grep, grepl, regexpr, gregexpr and regexec search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

txt <- "a test of capitalizing"  
gsub("(\\w)(\\w\*)", "\\U\\1\\L\\2", txt, perl=TRUE)

## [1] "A Test Of Capitalizing"

## 3. Regular Expression

Create a vector called mice with the following famous cartoon mice: “Mighty Mouse”, “Mickey Mouse”, “Speedy Gonzales”, “Stuart Little”, “Minnie Mouse”, “Gus Gus”, “Danger Mouse”

mice <- c("Mighty Mouse", "Mickey Mouse", "Speedy Gonzales", "Stuart Little", "Minnie Mouse", "Gus Gus", "Danger Mouse")  
  
for (i in mice) print(i)

## [1] "Mighty Mouse"  
## [1] "Mickey Mouse"  
## [1] "Speedy Gonzales"  
## [1] "Stuart Little"  
## [1] "Minnie Mouse"  
## [1] "Gus Gus"  
## [1] "Danger Mouse"

Use the str\_detect function to check for the string “Mouse”

for (i in mice) print(glue("{i}: {str\_detect(i, 'Mouse')}"))

## Mighty Mouse: TRUE  
## Mickey Mouse: TRUE  
## Speedy Gonzales: FALSE  
## Stuart Little: FALSE  
## Minnie Mouse: TRUE  
## Gus Gus: FALSE  
## Danger Mouse: TRUE

Use the str\_subset function to check for the string “Mouse”

str\_subset(mice, "Mouse")

## [1] "Mighty Mouse" "Mickey Mouse" "Minnie Mouse" "Danger Mouse"

Use the str\_subset function to check for the characters “Mi” at the beginning of the string. Force lower case.

str\_subset(tolower(mice), "^mi")

## [1] "mighty mouse" "mickey mouse" "minnie mouse"

Use the str\_subset function to check for the character “s” at the end of the string.

str\_subset(mice, "s$")

## [1] "Speedy Gonzales" "Gus Gus"

Use the str\_subset function to find Mickey or Minnie.

pattern <- c("Mickey", "Minnie")  
map(pattern, str\_subset, string = mice)

## [[1]]  
## [1] "Mickey Mouse"  
##   
## [[2]]  
## [1] "Minnie Mouse"