Stat 133 HW04: String Manipulation and Regex

Zhiqiang Liao, 24229382

Introduction

This assignment has two purposes:

a) to familiarize you with manipulating character strings
b) to introduce you to regular expressions in R

Submit your assignment to bcourses, specifically turn in your **Rmd** (R markdown) file as well as the produced pdf file. Make sure to change the argument `eval=TRUE` inside every testing code chunk.

Names of Files

Imagine that you need to generate the names of 4 data files (with .csv extension). All the files have the same prefix name but each of them has a different number: `file01.csv`, `file02.csv`, `file03.csv`, and `file04.csv`. We can generate a character vector with these names in R. One naive solution would be to write something like this:

```
files <- c('file01.csv', 'file02.csv', 'file03.csv', 'file04.csv')
```

Now imagine that you need to generate 100 file names. You could write a vector with 100 file names but it's going to take you a while.

How would you generate the corresponding character vector `files` in R containing 100 files names: `file01.csv, file02.csv, file03.csv, ..., file99.csv, file100.csv`? Notice that the numbers of the first 9 files start with 0.

```
# vector of file names
library(stringr)
sprintf("file%s.csv", str_pad(c(1:100), 2, "left", "0"))
```

```
##   [1] "file01.csv"  "file02.csv"  "file03.csv"  "file04.csv"  "file05.csv"
##   [6] "file06.csv"  "file07.csv"  "file08.csv"  "file09.csv"  "file10.csv"
##  [11] "file11.csv"  "file12.csv"  "file13.csv"  "file14.csv"  "file15.csv"
##  [16] "file16.csv"  "file17.csv"  "file18.csv"  "file19.csv"  "file20.csv"
##  [21] "file21.csv"  "file22.csv"  "file23.csv"  "file24.csv"  "file25.csv"
##  [26] "file26.csv"  "file27.csv"  "file28.csv"  "file29.csv"  "file30.csv"
##  [31] "file31.csv"  "file32.csv"  "file33.csv"  "file34.csv"  "file35.csv"
##  [36] "file36.csv"  "file37.csv"  "file38.csv"  "file39.csv"  "file40.csv"
##  [41] "file41.csv"  "file42.csv"  "file43.csv"  "file44.csv"  "file45.csv"
##  [46] "file46.csv"  "file47.csv"  "file48.csv"  "file49.csv"  "file50.csv"
##  [51] "file51.csv"  "file52.csv"  "file53.csv"  "file54.csv"  "file55.csv"
##  [56] "file56.csv"  "file57.csv"  "file58.csv"  "file59.csv"  "file60.csv"
##  [61] "file61.csv"  "file62.csv"  "file63.csv"  "file64.csv"  "file65.csv"
##  [66] "file66.csv"  "file67.csv"  "file68.csv"  "file69.csv"  "file70.csv"
```

```
##  [71] "file71.csv"  "file72.csv"  "file73.csv"  "file74.csv"  "file75.csv"
##  [76] "file76.csv"  "file77.csv"  "file78.csv"  "file79.csv"  "file80.csv"
##  [81] "file81.csv"  "file82.csv"  "file83.csv"  "file84.csv"  "file85.csv"
##  [86] "file86.csv"  "file87.csv"  "file88.csv"  "file89.csv"  "file90.csv"
##  [91] "file91.csv"  "file92.csv"  "file93.csv"  "file94.csv"  "file95.csv"
##  [96] "file96.csv"  "file97.csv"  "file98.csv"  "file99.csv"  "file100.csv"
```

USA States Names

One of the datasets that come in R is `USArrests`. The row names of this data correspond to the 50 states. We can create a vector `states` with the row names:

```
states <- rownames(USArrests)
head(states, n = 5)
```

```
## [1] "Alabama"    "Alaska"     "Arizona"    "Arkansas"   "California"
```

Use `nchar()` to answer the following questions:

- Obtain a frequency table with the number of characters of the states' names
- What are the states with the longest names?
- What are the states with the shortest names?
- What's the most common length of names (i.e. the mode)?

```
# your answers
table(nchar(states))
```

```
##
##  4  5  6  7  8  9 10 11 12 13 14
##  3  3  5  8 12  4  4  2  4  3  2
```

```
states[which.max(nchar(states))]
```

```
## [1] "North Carolina"
```

```
states[which.min(nchar(states))]
```

```
## [1] "Iowa"
```

```
max(table(nchar(states)))
```

```
## [1] 12
```

Using `grep()`

You can use the function `grep()` to know if there are any states containing the letter "z".

```r
# states containing the letter 'z'
grep(pattern = 'z', x = states)
```

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

In this case there is just one state (the third one) which corresponds to Arizona

You can also use `grep()` with its argument `value = TRUE` to obtain the value of the matched pattern:

```r
# states containing the letter 'z'
grep(pattern = 'z', x = states, value = TRUE)
```

```
## [1] "Arizona"
```

Your turn. Use `grep()`—and maybe other functions—to write the commands that answer the following questions:

- How many states contain the letter `i`?
- How many states contain the letter `q`?
- How many states do not contain the letter `a`?
- Which states contain the letter `j`?
- Which states contain the letter `x`?
- Which states are formed by two words?
- Which states start with `W` and end with a vowel?
- Which states start with `W` and end with a consonant?
- Which states contain at least three `i` (e.g. Illinois)?
- Which states contain five vowels (e.g. California)?
- Which states have three vowels next to each other (e.g. Hawaii)?

Tip: You can use `grep()`'s argument `ignore.case` to ignore letters in lower or upper case.

```r
# your answers
length(grep("i", x = states))
```

```
## [1] 25
```

```r
length(grep("q", x = states))
```

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

```r
length(grep("a", x = states, invert = TRUE))
```

```
## [1] 14
```

```r
states[grep("[jJ]", x = states)]
```

```
## [1] "New Jersey"
```

```
states[grep("[xX]", x = states)]
```

```
## [1] "New Mexico" "Texas"
```

```
states[grep(". .", x = states)]
```

```
##  [1] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
##  [5] "North Carolina" "North Dakota"   "Rhode Island"   "South Carolina"
##  [9] "South Dakota"   "West Virginia"
```

```
states[grep("^W.+[aeiou]$", x = states)]
```

```
## [1] "West Virginia"
```

```
states[grep("^W.+[^aeiou]$", x = states)]
```

```
## [1] "Washington" "Wisconsin"  "Wyoming"
```

```
states[grep("i.*i.*i.*", x = "iii", ignore.case = TRUE)]
```

```
## [1] "Alabama"
```

```
setdiff(states[grep("[aeiou].*[aeiou].*[aeiou].*[aeiou].*[aeiou]", x = states, ignore.case = TRUE)], sta
```

```
## [1] "California"     "North Carolina" "South Dakota"   "West Virginia"
```

```
states[grep("[aeiou][aeiou][aeiou]", x= states, ignore.case = TRUE)]
```

```
## [1] "Hawaii"    "Louisiana"
```

Starts with . . .

Write a function `starts_with()` such that, given a character string and a single character, it determines whether the string starts with the provided character.

```
# starts_with
starts_with = function(charString, singleChar){
  singleChar = str_c("^", singleChar, "")
  return (any(grepl(singleChar, x = charString)))
}
```

Test it:

```
starts_with("Hello", 'H')  # TRUE
```

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

```
starts_with("Good morning", 'H')  # FALSE
```

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

Ends with ...

Now write a function `ends_with()` such that, given a character string and a single character, it determines whether the string ends with the provided character.

```
# ends_with
ends_with = function(charString, singleChar){
  singleChar = str_c(singleChar, "$")
  return (any(grepl(singleChar, x = charString)))
}
```

Test it:

```
ends_with("Hello", 'o')  # TRUE
```

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

```
ends_with("Good morning", 'o')  # FALSE
```

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

Colors in Hexadecimal Notation

Write a function `is_hex()` that checks whether the input is a valid color in hexadecimal notation. Remember that a hex color starts with a hash `#` symbol followed by six hexadecimal digits: 0 to 9, and the first six letters A, B, C, D, E, F. Since R accepts hex-colors with lower case letters (a, b, c, d, e, f) your function should work with both upper and lower case letters.

```
# is_hex()
is_hex = function(input){
  return (grepl("#[0-9a-z]{6}", x = tolower(input)))
}
```

Test it:

```
is_hex("#FF00A7")  # TRUE
```

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

```
is_hex("#ff0000")  # TRUE
```

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

```
is_hex("#123456")  # TRUE
```

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

```
is_hex("#12Fb56")  # TRUE
```

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

```
is_hex("FF0000")   # FALSE
```

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

```
is_hex("#1234GF")  # FALSE
```

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

```
is_hex("#09892")   # FALSE
```

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

```
is_hex("blue")     # FALSE
```

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

Hexadecimal Colors with Transparency

Write a function `is_hex_alpha()` that determines whether the provided input is a hex color with alpha transparency. Remember that such a color has 8 hexadecimal digits instead of just 6.

```
# is_hex_alph()
is_hex_alpha = function(input){
  return (nchar(input) == 9)
}
```

Test it:

```
is_hex_alpha("#FF000078")  # TRUE
```

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

```
is_hex_alpha("#FF0000")   # FALSE
```

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

Splitting Characters

Create a function `split_chars()` that splits a character string into one single character elements.

```
# split_chars()
split_chars = function(input){
  return (unlist(strsplit(input, "")))
}
```

Test it:

```
split_chars('Go Bears!')
```

```
## [1] "G" "o" " " "B" "e" "a" "r" "s" "!"
```

```
split_chars('Expecto Patronum')
```

```
##  [1] "E" "x" "p" "e" "c" "t" "o" " " "P" "a" "t" "r" "o" "n" "u" "m"
```

Note that `split_chars()` returns the output in a single vector. Each element is a single character.

Number of Vowels

Create a function `num_vowels()` that returns the number of vowels of a character vector. In this case, the input is a vector in which each element is a single character.

```
# num_vowels()
num_vowels = function(x){
  x = paste(gsub("[^aeiou]", "", x), collapse = "")
  a = length(unlist(str_extract_all(x, "a")))
  e = length(unlist(str_extract_all(x, "e")))
  i = length(unlist(str_extract_all(x, "i")))
  o = length(unlist(str_extract_all(x, "o")))
  u = length(unlist(str_extract_all(x, "u")))
  output = c(a,e,i,o,u)
  names(output) = c("a","e","i","o","u")
  return (output)
}
```

Test it:

```
vec <- c('G', 'o', ' ', 'B', 'e', 'a', 'r', 's', '!')
num_vowels(vec)
```

```
## a e i o u
## 1 1 0 1 0
```

Notice that the output is a numeric vector with five elements. Each element has the name of the corresponding vowel.

Counting Vowels

Use the functions `split_chars()` and `num_vowels()` to write a function `count_vowels()` that computes the number of vowels of a character string:

```
# count_vowels()
count_vowels = function(x){
  return (num_vowels(split_chars(tolower(x))))
}
```

Test it:

```
count_vowels("The quick brown fox jumps over the lazy dog")
```

```
## a e i o u
## 1 3 1 4 2
```

Make sure that `count_vowels()` counts vowels in both lower and upper case letters:

```
count_vowels("THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG")
```

```
## a e i o u
## 1 3 1 4 2
```

Number of Consonants

Write a function `num_cons()` that counts the number of consonants regardless of whether thery are in upper or lower case (just the number, not the counts of each letter)

```
# num_cons()
num_cons = function(input){
  return (sum(table(str_extract_all(tolower(input), "[^aeiou[[:space:]]]"))))
}
```

Test it:

```
fox <- "The quick brown fox jumps over the lazy dog"
num_cons(fox)
```

```
## [1] 24
```

Reversing Characters

Write a function that reverses a string by characters

```
# reverse_chars()
reverse_chars = function(x){
  return (paste(rev(unlist(str_split(x, ""))),collapse=""))
}
```

Test it:

```
reverse_chars("gattaca")
```

```
## [1] "acattag"
```

```
reverse_chars("Lumox Maxima")
```

```
## [1] "amixaM xomuL"
```

Reversing Sentences by Words

Write a function `reverse_words()` that reverses a string (i.e. a sentence) by words

```
# reverse_words()
reverse_words = function(x){
  return (paste(rev(unlist(str_split(x, " "))), collapse = " "))
}
```

Test it:

```
reverse_words("sentence! this reverse")
```

```
## [1] "reverse this sentence!"
```

If the string is just one word then there's basically no reversing:

```
reverse_words("string")
```

```
## [1] "string"
```
