Stat 133 HW04 : String Manipulation and Regex

Hye Soo Choi

July 10, 2015

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
files <- paste('file', c(paste(0,1:9,sep=''), 10:100),'.csv', sep='')
```

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
nchar_states <- nchar(states)
 # frequency table with the number of characters of the states' name
table(nchar_states)
```

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

```
# What are the states with the longest names
states[nchar_states == max(nchar_states)]
```

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

```
# What are the states with the shortest names
states[nchar_states == min(nchar_states)]
```

```
## [1] "Iowa" "Ohio" "Utah"
```

```
#What's the most common length of names
range_nchar <- names(table(nchar_states))
range_nchar[max(table(nchar_states)) == table(nchar_states)]
```

```
## [1] "8"
```

Using `grep()`

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

```
# 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:

```
# 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.

```
# your answers

# How many states contain the letter ```i```?
length(grep('i', states, ignore.case = TRUE))
```

```
## [1] 28
```

```
# How many states contain the letter ```q```?
length(grep('q', states, ignore.case = TRUE))
```

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

```
# How many states do not contain the letter ```a```?
length(grep('^[^a]*$', states, ignore.case= TRUE))
```

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

```
# Which states contain the letter ```j```?
grep('j', states, value = TRUE, ignore.case= TRUE)
```

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

```
# Which states contain the letter ```x```?
grep('x', states, value =TRUE, ignore.case = TRUE)
```

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

```
# Which states are formed by two words?
grep(' ', states, value = TRUE)
```

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

```r
# Which states start with ```W``` and end with a vowel?
grep('^W.*[aeiou]$', states, value = TRUE)
```

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

```r
# Which states start with ```W``` and end with a consonant?
grep('^W.*[b-df-hj-np-tv-z]$', states, value = TRUE)
```

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

```r
# Which states contain at least three ```i``` (e.g. Illinois)?
grep('i.*i.*i', states, value = TRUE, ignore.case = TRUE)
```

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

```r
# Which states contain five vowels (e.g. California)?
grep('^[^aeiou]*[aeiou][^aeiou]*[aeiou][^aeiou]*[aeiou][^aeiou]*[aeiou][^aeiou]*[aeiou][^aeiou]*$',
     states, value = TRUE, ignore.case= TRUE)
```

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

```r
# Which states have three vowels next to each other (e.g. Hawaii)?
grep('[aeiou]{3}', states, value = TRUE, 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.

```r
# starts_with
starts_with <- function(str, char){
   length(grep(pattern = paste('^', char, sep = ''), x = str)) == 1
}
```

Test it:

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

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

```r
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(str, char){
  length(grep(pattern = paste(char, '$', sep = ''), x = str)) == 1
}
```

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(color){
    length(grep('^\\#[[:xdigit:]]{6}$', color , ignore.case = TRUE)) == 1
}
```

Test it:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```r
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.

```r
# is_hex_alpha()
is_hex_alpha <- function(color){
  length(grep('^\\#[[:xdigit:]]{8}$', color , ignore.case = TRUE)) == 1
}
```

Test it:

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

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

```r
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(str){
  vec <- character(0)
  for( k in 1:nchar(str)){
    vec[k] <- substr(str, k , k)
  }
  vec
}
```

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(vec){
  # number of 'a'
  num_a <- sum(tolower(vec) == 'a')
  num_e <- sum(tolower(vec) == 'e')
  num_i <- sum(tolower(vec) == 'i')
  num_o <- sum(tolower(vec) == 'o')
  num_u <- sum(tolower(vec) == 'u')
  c( a = num_a, e = num_e, i = num_i, o = num_o, u = num_u)
}
```

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(str){
  num_vowels(split_chars(str))
}
```

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(str){
  vowels <- c('a','e','i','o','u')
  consonants <- setdiff(letters, vowels)
  sum(tolower(split_chars(str)) %in% consonants)
}
```

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

```r
# reverse_chars()
reverse_chars <- function(str){
  split_str <- split_chars(str)
  reversed_vec <- split_str[length(split_str):1]
  paste(reversed_vec, collapse = '')
}
```

Test it:

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

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

```r
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

```r
# reverse_words()
reverse_words <- function(str){
  str_vec <- unlist(strsplit(str,  ' '))
  reversed_vec <- str_vec[length(str_vec):1]
  paste(reversed_vec, collapse = ' ')
}
```

Test it:

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

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

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

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

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