STAT 133 HW04: Strings Manipulation and Regex

Your name and SID

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: `plot01.png`, `plot02.png`, `plot03.png`, and `plot04.png`. We can generate a character vector with these names in R. One naive solution would be to write something like this:

```
files <- c('plot01.png', 'plot02.png', 'plot03.png', 'plot04.png')
```

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: `plot01.png, plot02.png, plot03.png, ..., plot99.png, plot100.png`? Notice that the numbers of the first 9 files start with 0.

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"
```

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.

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.

Here's an example: the string is `"Hello"` and we want to know if it starts with the letter `"H"`

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

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

In contrast, this other example returns `FALSE`

```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.

Here's an example:

```
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.

For instance:

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

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

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

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

Check your function with these values:

```
is_hex("#ff0000")  # TRUE
is_hex("#123456")  # TRUE
is_hex("#12Fb56")  # TRUE

is_hex("#1234GF")  # FALSE
is_hex("#1234567") # FALSE
is_hex("blue")     # 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.

For instance:

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

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

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

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

Hexadecimal Color split in RGB values

Write a function `hex_values()` that takes a hex-color and returns a vector with the values of the RGB—and possibly alpha—channels. For instance:

```r
# color with no transparency
hex_values("#435690")
```

```
##   red green  blue
##  "43"  "56"  "90"
```

If the provided color has an alpha channel, then the output should display such value:

```r
# color with transparency
hex_values("#435690FF")
```

```
##   red green  blue alpha
##  "43"  "56"  "90"  "FF"
```

If the input is not a valid hex-color, `hex_vales()` should return the message: `input is not a valid hexadecimal color`. The message should be displayed via `cat()`

```r
# invalid hex color
hex_values("#435XY90")
```

```
## input is not a valid hexadecimal color
```

Splitting Characters

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

For example:

```r
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.

For example:

```
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:

Here's what `count_vowels()` should do:

```
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)

For instance:

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

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

Reversing Characters

Write a function `reverse_chars()` that reverses a string by characters

For instance:

```
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

For example:

```
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"
```
