# MY607-Week03-MichaelY-textmanip

*Michael Y.*

*September 15, 2019*

## Contents

## ADCR, Chapter 8, Exercise 3 (p.217)

**Copy the introductory example. The vector *name* stores the extracted names.**

```
### Raw input data:
raw.data
```

```
## [1] "555-1239Moe Szyslak(636) 555-0113Burns, C. Montgomery555-6542Rev. Timothy Lovejoy555 8904Ned Fla
```

```
### extracted names (including titles and middle initials, as originally provided:
name <- unlist(str_extract_all(raw.data, "[[:alpha:]., ]{2,}"))
### for clarity, transpose to display as a column vector:
t(t(name))
```

```
##      [,1]
## [1,] "Moe Szyslak"
## [2,] "Burns, C. Montgomery"
## [3,] "Rev. Timothy Lovejoy"
## [4,] "Ned Flanders"
## [5,] "Simpson, Homer"
## [6,] "Dr. Julius Hibbert"
```

```
### extracted phone numbers (in format provided):
phone <- unlist(str_extract_all(raw.data, "\\(?(\\d{3})?\\)?(-| )?\\d{3}(-| )?\\d{4}"))
### for clarity, transpose to display as a column vector:
t(t(phone))
```

```
##      [,1]
## [1,] "555-1239"
## [2,] "(636) 555-0113"
## [3,] "555-6542"
## [4,] "555 8904"
## [5,] "636-555-3226"
## [6,] "5543642"
```

```
### Make a Data Frame:
name_phone_df = data.frame(name = name, phone = phone)
name_phone_df
```

```
##                   name          phone
## 1          Moe Szyslak       555-1239
## 2 Burns, C. Montgomery (636) 555-0113
## 3 Rev. Timothy Lovejoy       555-6542
## 4         Ned Flanders       555 8904
## 5       Simpson, Homer   636-555-3226
## 6   Dr. Julius Hibbert        5543642
```

**Use the tools of this chapter to rearrange the vector so that all elements conform to the standard first_name last_name.**

The above description is a bit unclear – I am unsure as to whether this means that titles should be stripped out.
Also, in the case of an individual like Mr. Burns, are we to use just the letter "C" as his first name?

I am going to make the assumption that the question is asking us to take those names which are presented as:

lastname, firstname

and rearrange such names so that the comma is removed and the names are presented in the order:

firstname lastname

but in the case of any titles, initials, etc., I am not going to remove them.

(The instructions should have been more clear if something different was desired.)

**STARTING NAMES:**

```
### for clarity, transpose to display as a column vector:
t(t(name))
```

```
##      [,1]
## [1,] "Moe Szyslak"
## [2,] "Burns, C. Montgomery"
## [3,] "Rev. Timothy Lovejoy"
## [4,] "Ned Flanders"
## [5,] "Simpson, Homer"
## [6,] "Dr. Julius Hibbert"
```

```
for (i in 1:length(name)) {
  if (str_detect(string = name[i], pattern = ",")) {
    origname <-unlist(str_split(string = name[i], pattern = ","))
    flippedname <- str_trim(paste(origname[2], origname[1]))
    name[i] <- flippedname
  }
}
```

**ENDING NAMES:**

```
### for clarity, transpose to display as a column vector:
t(t(name))
```

```
##      [,1]
## [1,] "Moe Szyslak"
## [2,] "C. Montgomery Burns"
## [3,] "Rev. Timothy Lovejoy"
## [4,] "Ned Flanders"
## [5,] "Homer Simpson"
## [6,] "Dr. Julius Hibbert"
```

Note that names #2 and #5 have been changed vs. original input.

**Construct a logical vector indicating whether a character has a title (i.e., Rev. and Dr.).**

```
hastitle = str_detect(string = name, pattern="^Rev\\.|^Dr\\.")
t(t(hastitle))
```

```
##       [,1]
## [1,] FALSE
## [2,] FALSE
## [3,]  TRUE
## [4,] FALSE
## [5,] FALSE
## [6,]  TRUE
```

**Display the names of those individuals with titles preceding their names**

```
t(t(name[hastitle]))
```

```
##      [,1]
## [1,] "Rev. Timothy Lovejoy"
## [2,] "Dr. Julius Hibbert"
```

**Construct a logical vector indicating whether a character has a second name.**

```
### How many parts does each name have?
numnameparts = unlist(lapply(X = str_split(name," "), FUN = length))
numnameparts
```

```
## [1] 2 3 3 2 2 3
```

```
cat(paste(numnameparts, name), sep="\n")
```

```
## 2 Moe Szyslak
## 3 C. Montgomery Burns
## 3 Rev. Timothy Lovejoy
## 2 Ned Flanders
## 2 Homer Simpson
## 3 Dr. Julius Hibbert
```

If the individual ***has*** a title (i.e.,. "Rev." or "Dr.", then his name must have ***4*** parts to include a middle name.
If the individual does ***not*** have a title, then his name must have ***3*** parts to include a middle name.

```
hasmiddlename = (hastitle & numnameparts == 4) | (!hastitle & numnameparts == 3)
t(t(hasmiddlename))
```

```
##       [,1]
## [1,] FALSE
## [2,]  TRUE
## [3,] FALSE
## [4,] FALSE
## [5,] FALSE
## [6,] FALSE
```

**Display the list of individuals who have a Middle Name:**

```
name[hasmiddlename]
```

```
## [1] "C. Montgomery Burns"
```

**If the instructions clearly specified to drop a title and/or middle name, then we could use the above to execute this.**

**In absence of such instructions, I'll leave the names as they are.**

---

## ADCR, Chapter 8, Exercise 4 (p.217)

**Describe the types of strings that conform to the following regular expressions and construct an example that is matched by the regular expression.**

### 1. [0-9]+\\$

This matches one or more digits, followed by a dollar-sign "$" character. (Note: It does **not** represent digits at the end of a line, because the backslashes preceding the dollar-sign give it its literal meaning, rather than its "end-of-line" meaning.)

```r
regex1 = '[0-9]+\\$'

a = '1$'
b = "2$"
c = '34$'
d = "56$"
e = 'xyz7$abc456$789'
f = "xyz89$abc$234$567"
g = 'xyz10$abc$asdf$'
h = "xyz11$abc$123456$asdfg"
x = "This is a long string with letters$9876 and 1234$numbers - many 56789$numbers"
example1 <- c(a,b,c,d,e,f,g,h,x)

unlist(str_extract_all(example1, regex(regex1)))
```

```
##  [1] "1$"      "2$"      "34$"     "56$"     "7$"      "456$"    "89$"     "234$"    "10$"
## [10] "11$"     "123456$" "1234$"   "56789$"
```

```r
##example1[grep(regex1, x=example1)]
unlist(regmatches(x=example1, m=gregexpr(text=example1, pattern=regex1)))
```

```
##  [1] "1$"      "2$"      "34$"     "56$"     "7$"      "456$"    "89$"     "234$"    "10$"
## [10] "11$"     "123456$" "1234$"   "56789$"
```

```r
for (i in list(a,b,c,d,e,f,g,h,x)){
  print(paste0(i,": "))
  print(t(t(unlist(str_extract_all(i,regex(regex1))))))
print("________________________")
}
```

```
## [1] "1$: "
##      [,1]
## [1,] "1$"
## [1] "________________________"
## [1] "2$: "
##      [,1]
## [1,] "2$"
## [1] "________________________"
## [1] "34$: "
##      [,1]
## [1,] "34$"
## [1] "______________________"
## [1] "56$: "
##      [,1]
## [1,] "56$"
## [1] "______________________"
## [1] "xyz7$abc456$789: "
##      [,1]
## [1,] "7$"
## [2,] "456$"
## [1] "______________________"
## [1] "xyz89$abc$234$567: "
##      [,1]
## [1,] "89$"
## [2,] "234$"
## [1] "______________________"
## [1] "xyz10$abc$asdf$: "
##      [,1]
## [1,] "10$"
## [1] "______________________"
## [1] "xyz11$abc$123456$asdfg: "
##      [,1]
## [1,] "11$"
## [2,] "123456$"
## [1] "______________________"
## [1] "This is a long string with letters$9876 and 1234$numbers - many 56789$numbers: "
##      [,1]
## [1,] "1234$"
## [2,] "56789$"
## [1] "______________________"
```

## 2. \\b[a-z]{1,4}\\b

Thus matches any "word" containing between 1 and 4 lower-case letters.
Note that in addition to spaces, a "word boundary" can also be delimited by certain characters such as a dollar-sign ("$") or a hyphen ("-").
Note however that an underscore ("_") is not considered a word boundary. Rather, an underscore is considered part of a word.

```
regex2 = "\\b[a-z]{1,4}\\b"
example2a = "This is a long sentence containing many words. Now$is$the$winter$of$our$discontent.  a$b."
example2b = "This.is.a.long.sentence.containing.many.words. Now-is-the-winter-of-our-discontent.  a-b."
example2c = "This:is:a:long:sentence:containing:many:words. Now_is_the_winter_of_our_discontent.  a_b."
res1=unlist(str_extract_all(example2a, regex(regex2)))
cat("\n____example2a_____", paste(1:length(res1),res1), "_________________\n",sep='\n')
```

```
##
## ____example2a_____
## 1 is
## 2 a
## 3 long
## 4 many
## 5 is
## 6 the
## 7 of
## 8 our
## 9 a
## 10 b
## _________________
```

```
res2=unlist(str_extract_all(example2b, regex(regex2)))
cat("\n____example2b_____", paste(1:length(res2),res2), "_________________\n",sep='\n')
```

```
##
## ____example2b_____
## 1 is
## 2 a
## 3 long
## 4 many
## 5 is
## 6 the
## 7 of
## 8 our
## 9 a
## 10 b
## _________________
```

```
#### the underscores do not constitute word separators, so the last portions will not be captured
res3=unlist(str_extract_all(example2c, regex(regex2)))
cat("\n____example2c_____", paste(1:length(res3),res3), "_________________\n",sep='\n')
```

```
##
## ____example2c_____
## 1 is
## 2 a
## 3 long
## 4 many
## __________________
```

Note that the underscores in example2c do not constitute word separators, thus none of the words are taken from the last part of that example.

**3. .*?\\.txt$**

This matches any number of characters (of any kind) followed by ".txt" , which must then conclude the line.

```r
regex3 = ".*?\\.txt$"
example3 = c(
  "This is a filename.txt",
  "Hello123456.txt",
  "what is new.txt",
  "This line will not be selected because .txt is not at the end.",
  "What if we put a linefeed following .txt\nAnd see what happens?"
)
result3=unlist(str_extract_all(example3, regex(regex3)))
result3
```

```
## [1] "This is a filename.txt" "Hello123456.txt"        "what is new.txt"
```

```r
cat(result3, sep="\n")
```

```
## This is a filename.txt
## Hello123456.txt
## what is new.txt
```

---

### 4. \\d{2}/\\d{2}/\\d{4}

This matches the usual way of displaying a date, e.g., mm/dd/yyyy (in the USA) or dd/mm/yyyy (elsewhere.)

```r
regex4 = "\\d{2}/\\d{2}/\\d{4}"
example4 = c(
"Today is 09/15/2019. Yesterday was 09/14/2019. But, Europeans write 14/09/2019.",
"Of course, there are no controls restricting months to be (01:12) or days to be (01:31)",
"So we could write 00/00/0000 or 99/99/9999 or 12/34/5678 and it would still be selected.",
"However, we can only use digits, not numbers, so we will not get ab/cd/efgh ."
)
unlist(str_extract_all(example4, regex(regex4)))
```

```
## [1] "09/15/2019" "09/14/2019" "14/09/2019" "00/00/0000" "99/99/9999" "12/34/5678"
```

---

**5. <(.+?)>.+?</\\1>**

This identifies an HTML or XML-style block where some tag references the start and end of such block, with ***non-empty*** text in between.
The start tag would be something like <FOO> and the corresponding end tag would be </FOO>.

```
regex5 = "<(.+?)>.+?</\\1>"
example5 = c(
  "<HTML>some content</HTML>",
  "<b>text content inside block</b>",
  "<bad>this will not work<bad>",
  "</bad2>also this will not work<bad2>",
  "<empty></empty>",
  "<notempty>x</notempty>",
  "</maybe>perhaps this will work?<//maybe>"
)
unlist(str_extract_all(example5, regex(regex5)))
```

```
## [1] "<HTML>some content</HTML>"                "<b>text content inside block</b>"
## [3] "<notempty>x</notempty>"                   "</maybe>perhaps this will work?<//maybe>"
```

## ADCR, Chapter 8, Exercise 9 (p.218)

**The following code hides a secret message. Crack it with R and regular expressions. Hint: Some of the characters are more revealing than others! The code snippet is also available in the materials at www.r-datacollection.com.**

```
### The code snippet with the secret message is here:
### http://www.r-datacollection.com/materials/regex/code_exercise.txt

#library(readr)        # loaded above

fileURL="http://www.r-datacollection.com/materials/regex/code_exercise.txt"
secret <- read_file(fileURL)
secret
```

```
## [1] "clcopCow1zmstc0d87wnkig7OvdicpNuggvhryn92Gjuwczi8hqrfpRxs5Aj5dwpn0TanwoUwisdij7Lj8kpf03AT5Idr3c
```

Let's see if any of the regex classes can help filter out something meaningful:

```
regexclasses <- c(
"[[:digit:]]",  #Digits: 0 1 2 3 4 5 6 7 8 9
"[[:lower:]]",  #Lowercase characters: a-z
"[[:upper:]]",  #Uppercase characters: A-Z
"[[:alpha:]]",  #Alphabetic characters: a-z and A-Z
"[[:alnum:]]",  #Digits and alphabetic characters
"[[:punct:]]",  #Punctuation characters: . , ; etc.
"[[:graph:]]",  #Graphical characters: [:alnum:] and [:punct:]
"[[:blank:]]",  #Blank characters: Space and tab
"[[:space:]]",  #Space characters: Space, tab, newline, and other space characters
"[[:print:]]"   #Printable characters: [:alnum:], [:punct:] and [:space:]
)

regexclasses
```

```
##  [1] "[[:digit:]]" "[[:lower:]]" "[[:upper:]]" "[[:alpha:]]" "[[:alnum:]]" "[[:punct:]]"
##  [7] "[[:graph:]]" "[[:blank:]]" "[[:space:]]" "[[:print:]]"
```

Let's try each regex class on the secret, and see if any of them gives insight:

```
resultlist = lapply(X=str_extract_all(string = secret, pattern=regexclasses),
                    FUN=str_c, collapse="")
unlist(resultlist)
```

```
## [1] "1087792855078035307553364116224905651724639589659490545"
## [2] "clcopowzmstcdwnkigvdicpuggvhrynjuwczihqrfpxsjdwpnanwowisdijjkpfdrcocbtyczjataootjtjnecfekrwwwoj
## [3] "CONGRATULATIONSYOUAREASUPERNERD"
## [4] "clcopCowzmstcdwnkigOvdicpNuggvhrynGjuwczihqrfpRxsAjdwpnTanwoUwisdijLjkpfATIdrcocbtyczjatOaootjt
## [5] "clcopCow1zmstc0d87wnkig7OvdicpNuggvhryn92Gjuwczi8hqrfpRxs5Aj5dwpn0TanwoUwisdij7Lj8kpf03AT5Idr3c
## [6] "....!"
## [7] "clcopCow1zmstc0d87wnkig7OvdicpNuggvhryn92Gjuwczi8hqrfpRxs5Aj5dwpn0TanwoUwisdij7Lj8kpf03AT5Idr3c
## [8] "\n"
## [9] "clcopCow1zmstc0d87wnkig7OvdicpNuggvhryn92Gjuwczi8hqrfpRxs5Aj5dwpn0TanwoUwisdij7Lj8kpf03AT5Idr3c
```

The third one shows promise – it has some discernable words, but they are run together.

```
regexclasses[3]
```

```
## [1] "[[:upper:]]"
```

```
resultlist[[3]]
```

```
## [1] "CONGRATULATIONSYOUAREASUPERNERD"
```

I see some punctuation in the sixth item:

```
regexclasses[6]
```

```
## [1] "[[:punct:]]"
```

```
resultlist[[6]]
```

```
## [1] "....!"
```

Perhaps we should try combining both regexes (i.e., 3 and 6) ?

```
combinedregex=sub(pattern="\\]\\[", replacement="", x = paste0(regexclasses[3],regexclasses[6]))
combinedregex
```

```
## [1] "[[:upper:][:punct:]]"
```

Here's the result from combining the UPPERCASE and the punctuation:

```
result9=str_c(unlist(str_extract_all(string = secret, pattern=combinedregex)), collapse="")
result9
```

```
## [1] "CONGRATULATIONS.YOU.ARE.A.SUPERNERD!"
```

OK, time to clean up the punctuation. Since the original secret string didn't contain any spaces, clearly it used a dot in place of a space. So, let's switch them back:

```
result9a = str_replace_all(result9,"\\."," ")
result9a
```

```
## [1] "CONGRATULATIONS YOU ARE A SUPERNERD!"
```

To make this grammatically correct, it needs a comma:

```
result9b = str_replace(result9a," ",", ")
result9b
```

```
## [1] "CONGRATULATIONS, YOU ARE A SUPERNERD!"
```

**The secret message is: "CONGRATULATIONS, YOU ARE A SUPERNERD!"**

**(Please, tell me something I *didn't* already know?)**