“First programming exercise”

Alan Berger Aug 20, 2020 minor edits Jan 18, 2021

additional R code, run the functions, and some text edits Aug 22, 2020

Introduction

This is the first in a sequence of programming exercises intended to fill the gap between learning the correct syntax of basic R commands and the programming assignments in the R Programming course in the Johns Hopkins University Data Science Specialization on Coursera. In this sequence of exercises in “composing” an R function to carry out a particular task, the idea is to practice correct use of R constructs and built in functions (functions the “come with” the basic R installation), while learning how to “put together” a correct sequence of blocks of commands that will obtain the desired result.

In these exercises, there will be a statement of what your function should do (what are the input variables and what the function should return) and a sequence of “hints”. To get the most out of these exercises, try to write your function using as few hints as possible.
Note there are often several ways to write a function that will obtain the correct result. For these exercises the directions and hints may point toward a particular approach intended to practice particular constructs in R and a particular line of reasoning.
There may be an existing R function or package that will do what is stated for a given practice exercise, but here (unlike other aspects of the R Programming course) the point is to practice formulating a logical sequence of steps, with each step a section of code, to obtain a working function, not to find an existing solution or a quick solution using a more powerful R construct that is better addressed later on.

Motivation for this exercise

The input data for many functions will be or include a tab delimited text file that comes from or is naturally viewed as an Excel spreadsheet. For example, the last programming assignment in the R Programming course uses a .csv (comma separated variables) text file appropriately read in using R’s read.csv function. It has a total of 46 columns, only 5 of which are relevant to the assignment (Excel columns B, G, K, Q and W). For this and many other situations, it would be convenient to have an R function that would take as its input an Excel column name (as a character variable) and output the corresponding column number.

Instruction for this function

For the first version of your function, just have it work for a column letter between a and z and only take as input a lower case letter. The skeleton of your function should “look like”

```
colnameToNumber <- function(colname) {
# convert an Excel column name
# to the corresponding column number;
# colname should be a lower case letter in the
# form of a character string, for example "a" or "r"
# "between" a and z (including a and z)

#   coding lines

    return(colnumber)
}
```

Directions: do NOT use a whole bunch of if statements (yes, one could do 26 if statements of the form `if(colname == "a") colnumber = 1` and so on, but, among other things, it would be easy to have typographic mistakes, and writing all those out would get pretty boring).

Note that the built in R variable (R object) **letters** is the character vector with entries “a”, “b”, . . . , “z” and note the built in R function **which** takes as input a logical vector (often defined by whether some condition is or is not true) and outputs the integer vector of the *indices* of the input vector for which the logical value is **TRUE**, so for example `which(1:4 < 3)` is equal `c(1, 2)`, and `which(sqrt(1:4) == 2)` is the single value 4 (since `sqrt(1:4)` is the vector c(1, $\sqrt{2}$, $\sqrt{3}$, 2) which is `c(1, 1.414214, 1.732051, 2)` (to the number of digits printed).

Your function, when applied to b, g, k, q and w (as character variables) should return 2, 7, 11, 17 and 23, respectively, and `colnameToNumber("zz")` should “throw an error” along the lines of

Error in colnameToNumber(“zz”) : no match to colname

This comes from the R statement: `stop("no match for colname")` which you should have occur in your function if it is the case that the input was not a lower case letter between a and z.

Try writing your program now before going to the additional hints below (the more you do “on your own” the faster you will gain skill at programming).

Further hints

You can use the **which** function to “pick out” the index k (an integer) of the **letters** vector for which `letters[k]` equals colname, this is the number you want to return.

How might you check for whether colname was a letter between “a” and “z”? What will the **which** function return if colname was not a “valid” value for the colnameToNumber function as currently constructed (colname is not equal any entry of **letters**)? (Try it out in an R session.)

Things to think about: how would you “extend” your function to treat some additional Excel columns, say through column “dz” (check out the **paste** function, you could use it along with **letters** to construct vectors containing additional column names and concatenate them together). You could do this with several lines of code to get the vector of “a” through “dz” (a through z; aa through az, ba through bz; ca through cz; da through dz; then concatenate them into 1 vector). To be able to find the column number for many more Excel columns you would want to use a for loop to construct and concatenate blocks of column names.

How would you extend your function to easily deal with upper case letters or a “mix” of upper and lower case letters, as in the column name “Cz” (some people like me are sloppy typists, or will forget about a lower case restriction and use capital letters as Excel does).
Now is the time to make use of Google to search for a built in R function that will convert any upper case letters in a character variable to lower case (and leave other characters as is).

How would you extend your function to treat a vector (of length > 1) of column names?

A working version of this function is given below, but try to do your own function before looking.

colnameToNumber function

```r
colnameToNumber <- function(colname){
# convert an Excel column name to the
# corresponding column number;
# should not depend on whether the
# letter(s) in colname are upper or
# lower case or a mixture of upper and
# lower case
# colname should be a character string such as "k" or "dz"

    colname <- tolower(colname)
# so only need to deal with lower case

# arrange to treat column names from "a" up through "dz"
    az <- letters  # c("a", ... , "z")
    aaz <- paste("a", az, sep = "")
# c("aa", "ab", ... , "az")
# additional column names
    baz <- paste("b", az, sep = "")
    caz <- paste("c", az, sep = "")
    daz <- paste("d", az, sep = "")

# concatenate the column names
    colnames <- c(az, aaz, baz, caz, daz)

# get the number corresponding to the input column name
# using the which function
    colnumber <- which(colnames == colname)
# test for having found a (unique) match
    if(length(colnumber) != 1) stop("no match to colname")
    return(colnumber)
}
```

Do check runs.

```r
colnameToNumber("a")
```

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

```r
colnameToNumber("z")
```

```
## [1] 26
```

```r
colnameToNumber("aa")
```

```
## [1] 27
```

```r
colnameToNumber("az")
```

```
## [1] 52
```

```r
colnameToNumber("ba")
```

```
## [1] 53
```

```r
colnameToNumber("bz")
```

```
## [1] 78
```

```r
colnameToNumber("ca")
```

```
## [1] 79
```

```r
colnameToNumber("cz")
```

```
## [1] 104
```

```r
colnameToNumber("da")
```

```
## [1] 105
```

```r
colnameToNumber("dz")
```

```
## [1] 130
```

```r
# colnameToNumber("zz") # should get error message
```

Note because we used the built in **letters** character vector, the calls to colnameToNumber above should be sufficient to check that it has been coded correctly. Contrast that with the checks that would be needed if we had used if tests for each column name between a and dz.

For the last part of this exercise, write a function that uses the colnameToNumber function to convert a character *vector* of column names to the corresponding vector of column numbers. One could modify the colnameToNumber function to do this using a for loop, but here write a separate function that uses the colnameToNumber function (this set up will be used as practice with the **sapply** function later on).

Hints: The output of this function, call the function colvecToNumbers will be an integer vector, call it colnumbers, having the same **length** as the input character vector, call it colvec. One can initialize colnumbers using the **integer**
function, and then "fill in" its entries in a for loop using the colnameToNumber function.

A working version of this function is given below, but try to do your own function before looking.

colvecToNumbers function

```r
colvecToNumbers <- function(colvec){
# convert a character vector of Excel column names
# to the integer vector of corresponding column numbers
# each entry of colvec should be a column name between a and dz

# check the input is a non-empty character vector
# note a single character string is a character vector of length 1
if(length(colvec) < 1) stop("colvec input to colvecToNumbers is empty")
if(!is.character(colvec)) stop("colvec input to colvecToNumbers is not character")

# create the integer vector to be output
    n <- length(colvec)
    colnumbers <- integer(n)
# use colnameToNumber to get each entry of colnumbers
    for (i in 1:n) {
        colname <- colvec[i]
        colnumbers[i] <- colnameToNumber(colname)
    }

return(colnumbers)
}
```

check runs for colvecToNumbers

```
colvec <- c("b", "g", "k", "q", "w") # the hospital data relevant columns
colvecToNumbers(colvec)
```

```
## [1]  2  7 11 17 23
```

```
colvec <- c("a", "aa", "ba", "ca", "da") # start of groups of 26 columns
colvecToNumbers(colvec)
```

```
## [1]   1  27  53  79 105
```

```
colvec <- c("z", "az", "bz", "cz", "dz") # end of groups of 26 columns
colvecToNumbers(colvec)
```

```
## [1]  26  52  78 104 130
```

Agrees with what it should be.

Hope this programming exercise was informative and good practice with writing a function.

Things to look forward to: later in the R Programming course one will learn the "apply family" of powerful built in R functions. Not counting the lines that check whether the input value of colvec is valid, one can write a version of colvecToNumbers using the one!! line

sapply(colvec, colnameToNumber)

Nice and concise and only one line needs to be checked for any mistakes - **sapply** does the for loop and creates the vector to be returned without any more lines of code needed. Fewer lines of code (as long as they are reasonably "readable" - not a birds nest of parentheses and brackets) mean fewer chances to make a mistake and fewer places for bugs to hide.