Seventh R function: systematic debugging; practice composing a function and constructing a data frame

Alan E. Berger January 25, 2020

available at https://github.com/AlanBerger/Practice-programming-exercises-for-R

1. Listing files that match any entry of search.strings; 2. Systematic Debugging

Introduction

This is the seventh in a sequence of programming exercises in "composing" an R function to carry out a particular task. Several of these "exercise files" likely
will take several sessions to master the content. The material below practices composing a logical sequence of steps to program a function that will accomplish a specified task, and preparing a corresponding data frame. It also introduces a systematic way to debug a program.

The idea of this set of exercises is to practice correct use of R constructs and built in functions (functions that "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.
Note these exercises are quite cumulative - one should do them in order.

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, even if there is a more efficent way to obtain the same result.
There may also be an existing R function or package that will do what is stated for a given practice exercise, but here 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

In my hands, searching for files that have one or more specified text strings within the file name using
my computer's search facility has not always given back what I thought it should. The function I constructed in R to do this is is the basis for the previous set of exercises and for this one.

I will also introduce an effective way to systematically debug a program. R has "software" that is very helpful in debugging, which is covered later in the R Programming course in the Johns Hopkins University Data Science Specialization on Coursera, but the elementary approach of "viewing' variables as they are defined or changed works quite well for moderate size functions as illustrated below. This is quite sufficient for debugging all the functions in the assignments for the R Programming class.

In the previous (sixth) exercise file we prepared the function

search_for_filenames_containing_multiple_patterns_and_output_file_info

the final version of which is copied here:

```r
search_for_filenames_containing_multiple_patterns_and_output_file_info <-
                                             function(directory, search.strings){

# directory is an absolute path (full path) or a path relative to the R working directory to the
# folder to be searched. If want to search the R working directory itself,
# can set directory = "."
# or could set directory to be the full path to the R working directory.

# search.strings is a character string vector

# Return a data frame of the file names (not including folders) in directory that contain
# ALL the entries of the search.strings vector somewhere in their file name,
# the search will be case insensitive (treats lower case and upper case letters as the same).

# Use R's list.files function to list all the files (file names) in directory that contsin
# search.strings[1] in their name, then eliminate names of folders, then use R's
# grep function to find out, for each file name, whether or not each
# character string in search.strings appears in the file name.
# For the files whose names contain all the character strings in search.strings, use
# R's file.info function to get the file size and last modification time.

# The output data frame will contain the file size and the last modification time
# for each file that has each member of search.strings somewhere in its file name.
#
# We will get the file names without the folder path leading to the files included in the name.

# check that search.strings is a non-empty character vector

ns <- length(search.strings)
if(ns < 1) stop("no entries in search.strings")
if(!is.character(search.strings)) stop("search.strings is not a character vector")

# We will return a data frame whose first column contains the files in directory
# that contain each character string in the search.strings vector somewhere in their name.
# The second column will contain the last time (and date) the file was modified,
# and the third column will be the file size (in bytes).

# The first step is to get all the file names in directory that contain the first string in
# search.strings somewhere in their file name; do not include the path
# to the file in the file name.

filenames <- list.files(directory, pattern = search.strings[1],
                        full.names = FALSE, ignore.case = TRUE)

# If at any point there are no files then return a message that there are none

if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# Eliminate names of any folders (directories) that are in the filenames character vector.
# paste(directory, "/", filenames, sep = "") will get the filenames including either
# the relative path from the R working directory or the absolute path (depending on what
# directory is); use these names in the R dir.exists function to check for
# folder (directory) names in filenames
```

```r
filenames <- filenames[!dir.exists(paste(directory, "/", filenames, sep = ""))]
# exclude directory names

if(length(filenames) == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# if there is more than 1 search string, search over the rest of them
if(ns > 1) {
   for(k in 2:ns) {
   filenames <- grep(search.strings[k], filenames, ignore.case = TRUE, value = TRUE)
#                grep returns the subset of filenames that contain
#                search.strings[k] in the file name
#                if filenames is an empty character vector, grep will just return
#                an empty character vector
   }
}

nf <- length(filenames)
if(nf == 0) {
   print("no files contain all the search strings")
   return("no files contain all the search strings")
}

# If got to here, at least 1 file has all the character strings in search.strings
# in its name, so get the information on these file(s) into a data frame.

################################## get the data frame to be output

# Get the desired output data frame using vectors

dfcolnames <- c("file.name", "modif.date", "size.in.bytes")
# initialize the 3 vectors that will hold this information on the files
# whose names matched all the members of search.strings

fname <- character(0)
fdate <- character(0)
fsize <- numeric(0)

for(k in 1:nf) {
    finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
# needed to include the path to the file so file.info can locate it
    fname <- c(fname, filenames[k])
    fdate <- c(fdate, as.character(finfo$mtime))
    fsize <- c(fsize, finfo$size)
}

df <- data.frame(fname, fdate, fsize, stringsAsFactors = FALSE)
colnames(df) <- dfcolnames

################################## finished getting the data frame to be output
```

```
# Write the data frame out to a tab delimited text file called scrlisting.txt in directory
# (i.e., in the folder specified by the argument directory this function was called with).

outpfilename <- paste(directory, "/", "scrlisting.txt", sep = "")
# One can rename this "scratch file" as desired after viewing it (best viewed in Excel).
write.table(df, file = outpfilename,
            append = FALSE, quote = FALSE, sep = "\t",
            row.names = FALSE, col.names = TRUE)
# This call to write.table will write out a data frame
# as one would usually want; it specifies the column separator to be a tab

return(df)
}
```

How the function **search__for__filenames__containing__multiple__patterns__and__output__file__info** works

From the previous set of exercises, these are the steps in composing this search function: After checking that search.strings is a non-empty character vector, list.files is used to get a character vector of the file and folder names that contain (match) the first entry of **search.strings** Then any names of directories are eliminated. Then for each of the rest of the entries of search.strings (if there is more than 1 entry) the grep function is used to retain only the file names containing that entry of search.strings After all the entries of search.strings are checked, one has a vector of the desired file names. If no files remain, the function exits with a message to the effect that there were no files matchng all the entries of search.strings If there were file(s) that matched all the entries, a data frame, df, is constructed containing the file names in column 1. The **file.info** function is used to find the last modification date for each of these files (placed in column 2 of df), and the file size in bytes (placed in column 3 of df). The data frame df is then written to a file located in directory using write.table, and then df is returned.

Exercises

The function for this exercise will be to construct a modified version of the search function above called

search__for__filenames__containing__any__of__the__patterns__and__output__file__info(directory, search.strings)

that will return the vector of file names in directory that match **ANY entry** of search.strings (i.e., 1 or more entries of search.strings), **rather than ALL the entries** of search.strings One way to view this, is that the previous function obtained the **intersection** of the sets of file names matching each entry of search strings, while this function is to obtain the **union** U of the sets of file names matching each entry of search strings. The next programming exercise will be to construct a modified version of this function that will return the vector of file names in directory that **DO NOT match ANY** entry of search.strings. This can be viewed as obtaining, relative to the set of all the files in directory, the **complement of U**.

Test runs on my computer

Recalling from the previous exercise file, I have constructed in my R working directory a folder called test_dir containing a small number of files with names I picked to conveniently test the **search__for__filenames__containing__multiple__patterns__and__output__file__info** function. Here are all the file names (and the one folder name) in the test_dir folder:

```
directory <- "test_dir"
list.files(directory)  # 9 files and 1 folder
```

```
 [1] "001.csv"         "002.csv"         "003.txt"
 [4] "004csvfile.txt"  "005txt2csv"      "1.csv"
 [7] "308.csv"         "folder001txtcsv" "scrlisting.txt"
[10] "txt.csv"
```

We will use this folder and files (9 filenames and 1 folder name) to test the search function to be written below (you can construct a similar test_dir folder and files with these filenames and also a folder in it called "folder001txtcsv" to run tests, the only difference will be the dates and sizes of the files).

programming exercise

Write a modification of the function immediately above, now called

```
search_for_filenames_matching_any_of_the_patterns_and_output_file_info <-
                                               function(directory, search.strings){
```

which returns information on all the files in directory that contain (match) **ANY** (i.e., 1 or more) of the text strings given in search.strings.

Hints: start by initializing

```
filenames <- character(0)
```

and then, in a for loop over the entries S of search.strings, find the vector V of file names that match S (and eliminate any names of folders) and append each V to filenames. After this is done, the R **unique** function can be used to eliminate duplicate entries in filenames - you will want to use this since a file might have matches to more than one entry of search.strings and we only want to list it once in the output. An example of what the unique function does is:

```
unique(c(1,2,3,3,2,1))
[1] 1 2 3
```

For this function (to find file names that match **any** entry of search.strings), one can modify the function above (which finds file names that match **all** the entries of search.strings). All the modifications will be **above** the line

```
################################### get the data frame to be output
```

Try doing this - a version is given below.

Hints: for this function it is natural to use the approach of appending additional file names in a vector V to an existing vector of file names via `filenames <- c(filenames, V)` Here in place of using grep we can repeatedly use list.files to get the file names that match each entry of search.strings, eliminate any folder names, and then append them. After this is done use the unique function to remove duplicate names before constructing the data frame to be output.

Here is a version that works (**except for 2 "oversights"**) that I will use as an example for how to systematically debug a function:

```
search_for_filenames_matching_any_of_the_patterns_and_output_file_info <-
                                          function(directory, search.strings){

# there are 2 errors in this function, for the purpose of demonstrating debugging
# one of the errors has been done below this line:
################################## get the data frame to be output

# directory is an absolute path (full path) or a path relative to the R working directory to the
# folder to be searched. If want to search the R working directory itself,
# can set directory = "." with the line of code:   directory <- "."
# or could set directory to be the full path to the R working directory.

# search.strings is a character string vector

# Return a data frame of the file names (not including folders) in directory that contain
# ANY of the entries of the search.strings vector somewhere in their file name,
# (or at the beginning of the filename, or at the end of the filename, if so specified).
# The search will be case insensitive (treats lower case and upper case letters as the same).

# The first step is to initialize the filenames vector: filenames <- character(0)

# In a for loop, use R's list.files function to list all the files (file names) matching the
# entries of search.strings, one by one,
# eliminating names of folders, and appending them to filenames

# Use the unique function to eliminate duplicates (keep only 1 copy of each file name)

# For the files whose names contain any of the character strings in search.strings, use
# R's file.info function to get the file size and last modification time as in the previous function.

# The output data frame will contain the file size and the last modification time
# for each file that has any member of search.strings somewhere in its file name
# (or at the beginning or at the end of the file name, if so specified)
#
# We will get the file names without the folder path leading to the files included in the name.

# check that search.strings is a non-empty character vector

ns <- length(search.strings)
if(ns < 1) stop("no entries in search.strings")
if(!is.character(search.strings)) stop("search.strings is not a character vector")

# We will return a data frame whose first column contains the files in directory
# that contain any character string in the search.strings vector somewhere in their name.
# The second column will contain the last time (and date) the file was modified,
# and the third column will be the file size (in bytes).

# The first step is to initialize the filenames vector
filenames <- character(0)

# Then in a for loop, for each entry S of search strings,
# use list.files to get the vector V the file names in directory that
# contain S in their file name and append V to filenames (after eliminating any folder names).
```

```r
# do not include the path to the file in the file name.

# To eliminate names of any folders (directories) that are in V:
# do paste(directory, "/", V, sep = "") to get the filenames including either
# the relative path from the R working directory or the absolute path (depending on what
# directory is); use these names in the R dir.exists function to check for folder
# (directory) names in filenames

   for (k in 1:ns) {
      V <- list.files(directory, pattern = search.strings[k],
                      full.names = FALSE, ignore.case = TRUE)
#     exclude directory names from V (we need to do this since we are "adding"
#     the file names in V to filenames and want only file names, not folder names
#     if V is empty (character(0)) then skip this
      if(length(V) > 0) V <- V[!dir.exists(paste(directory, "/", V, sep = ""))]
      filenames <- c(filenames, V)
   }

### It is important to note we could have also "run the for loop" in this fashion:
### for (S in search.strings) {
###    V <- list.files(directory, pattern = S,
##############   rest of the for loop
###

nf <- length(filenames)
if(nf == 0) {
   print("no files contain any of the search strings")
   return("no files contain any of the search strings")
}

filenames <- unique(filenames)

# If got to here, at least 1 file has a character string in search.strings
# in its name, so get the information on these file(s) into a data frame.

################################### get the data frame to be output

# Get the desired output data frame using vectors

dfcolnames <- c("file.name", "modif.date", "size.in.bytes")
# initialize the 3 vectors that will hold this information on the files
# whose names matched any of the members of search.strings

fname <- character(0)
fdate <- character(0)
fsize <- numeric(0)

for(k in 1:nf) {
    finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
# needed to include the path to the file so file.info can locate it
    fname <- c(fname, filenames[k])
    fdate <- c(fdate, as.character(finfo$mtime))
    fsize <- c(finfo$size)
}

df <- data.frame(fname, fdate, fsize, stringsAsFactors = FALSE)
colnames(df) <- dfcolnames

################################## finished getting the data frame to be output

# Write the data frame out to a tab delimited text file called scrlisting.txt in directory
# (i.e., in the folder specified by the argument directory this function was called with).

outpfilename <- paste(directory, "/", "scrlisting.txt", sep = "")
# One can rename this "scratch file" as desired after viewing it (best viewed in Excel or equivalent).
write.table(df, file = outpfilename,
            append = FALSE, quote = FALSE, sep = "\t",
            row.names = FALSE, col.names = TRUE)
# This call to write.table will write out a data frame
# as one would usually want; it specifies the column separator to be a tab

return(df)
}

# do test runs on your computer
```

Here are a couple test runs on my computer of this version of

```
search_for_filenames_matching_any_of_the_patterns_and_output_file_info
```

test runs for this search function (which has 2 errors) listing file names matching any entry in search.strings

```
# test runs on my computer
directory <- "test_dir"
# note the  scrlisting.txt  file contains the output from
# the previous run of the search function so its modification date and file size will vary
list.files(directory)
 [1] "001.csv"         "002.csv"         "003.txt"
 [4] "004csvfile.txt"  "005txt2csv"      "1.csv"
 [7] "308.csv"         "folder001txtcsv" "scrlisting.txt"
[10] "txt.csv"

search.strings <- c("001", "30")  # should NOT get the folder name folder001txtcsv
search_for_filenames_matching_any_of_the_patterns_and_output_file_info(directory, search.strings)
  file.name          modif.date size.in.bytes
1   001.csv 2012-09-21 16:00:20         74305
2   308.csv 2012-09-21 16:00:20         74305

# ******   correctly does not include the folder,
# ******   but the correct file size for 001.csv 31271 bytes     ******

search.strings <- c("1234", "2222")
search_for_filenames_matching_any_of_the_patterns_and_output_file_info(directory, search.strings)
[1] "no files contain any of the search strings"
```

```
[1] "no files contain any of the search strings"

# as expected


search.strings <- c("001", "005", "txt2")
search_for_filenames_matching_any_of_the_patterns_and_output_file_info(directory, search.strings)
   file.name          modif.date size.in.bytes
1    001.csv 2012-09-21 16:00:20            NA
2 005txt2csv 2020-12-13 10:03:14            NA
3       <NA>                <NA>            NA

# How did this happen? The first two lines have the correct file names and date,
# but what accounts for the third output line with all NA's?, and all NA's for the file sizes?
```

Debugging is an essential skill in doing programming. For "simple" **syntax errors** such as not having a required parenthesis or square or curly bracket or having a left one when a right one is needed etc., R will generally stop with an error message that even if not compleely clear, will usually give a good indication of what and where to look for the cause of the error. Debugging "logical errors" - where the code runs but does not do what you wanted, or worse, where the code does what you intended it to do, but that does not give a correct solution for the task at hand, can be much more challenging. Here there is 1 file name with "001" in it; and the same file (005txt2csv, and no others) has "005" and "txt2" in its name. We used the **unique** function to eliminate duplicate file names, so what went wrong? One might well look at the errors and look over parts the code that seem related to the incorrect behavior - that is a good way to start - and often leads directly to a solution. When that doesn't work, the following approach can usually uncover the problem(s).

A systematic way to debug

A way to see "what is going on" for debugging (and also very useful as one is writing a function) is to run the lines of the function one by one (or a few at a time) in the R console, NOT within the function. First set values for the input arguments of the function using choices that will run a short test case,
and then go through the lines of the function as illustrated below. Every time a variable (R object) is defined or changed "look at" its value and see if that is what you intended and if that is getting the desired result. For single variables (length 1 vectors) or short vectors look at the RStudio Environment-Global Environment sub-window which lists R objects in the R workspace and their value, or for larger vectors or data frames, information on them. Have R display the value of small vectors or data frames by typing the name of the variable on one line
("small" meaning not cluttering the console with "too much" output).
For larger objects use **head** and/or **tail** or **str** (structure) to get information. For a large list, perhaps do str on the first and last items in the list. With for loops, run the loop "by hand", setting the index of the loop equal the first item and running the lines of the loop one by one; then set the index of the loop equal the second item and run the lines of the loop, etc. This is where a good choice of test case to run is important - choosing input arguments of the function so the run "by hand" will be manageable. This approach will usually uncover what is wrong with a small or moderate size function. This is also another reason why it is best to "split up" a "large" function into separate "pieces / modules" (separate functions) that can individually be tested and debugged. Such individual functions can sometimes be reused (or reused after slight modification) and give more flexibility in constructing complicated functions.

Here is an example, systematically uncovering the two errors in the function above. I have added comment notes on the line by line run of the function above. Even if you have already spotted the errors, it is worth looking over the example to see how this works.

```
directory <- "test_dir"
```

```
search.strings <- c("001", "005", "txt2")  # this choice will reveal both errors
# search.strings was chosen to have just 2 (unique) matching files, and one of the files
# matches 2 of the search strings, so this gives a "small" test case that utilizes
# most of the features of the code

# run the lines of
# search_for_filenames_matching_any_of_the_patterns_and_output_file_info
# line by line in the R console

# skip the comment lines

ns <- length(search.strings)
# one can see in the Global Environment sub-window
# of RStudio that ns is 3L (3 as an integer)
if(ns < 1) stop("no entries in search.strings")
if(!is.character(search.strings)) stop("search.strings is not a character vector")
# search.strings is acceptable

filenames <- character(0)
# an empty character initial value for filenames

# RUN THE FOR LOOP BY HAND 1 INDEX VALUE AT A TIME

k <- 1
V <- list.files(directory, pattern = search.strings[k],
                full.names = FALSE, ignore.case = TRUE)

# display V (also can read it in the Global Environment sub-window)
V
[1] "001.csv"         "folder001txtcsv"

if(length(V) > 0) V <- V[!dir.exists(paste(directory, "/", V, sep = ""))]
V
[1] "001.csv"
# the directory name was excluded correctly

# append V to filenames
filenames <- c(filenames, V)
filenames
[1] "001.csv"
############ end of loop for k = 1

# run the loop for k = 2
k <- 2
V <- list.files(directory, pattern = search.strings[k],
                full.names = FALSE, ignore.case = TRUE)

# display V (also can read in the Global Environment sub-window)
V
[1] "005txt2csv"

if(length(V) > 0) V <- V[!dir.exists(paste(directory, "/", V, sep = ""))]
V
[1] "005txt2csv"
```

```
# append V to filenames
filenames <- c(filenames, V)
> filenames
[1] "001.csv"    "005txt2csv"
############ end of loop for k = 2

# run the loop for k = 3
k <- 3

V <- list.files(directory, pattern = search.strings[k],
                full.names = FALSE, ignore.case = TRUE)

# display V (also can read in the Global Environment sub-window)
V
[1] "005txt2csv"

if(length(V) > 0) V <- V[!dir.exists(paste(directory, "/", V, sep = ""))]
V
[1] "005txt2csv"

# append V to filenames
filenames <- c(filenames, V)
filenames
[1] "001.csv"    "005txt2csv" "005txt2csv"

############ end of loop for k = 3
# finished doing the for loop   for (k in 1:ns){   by hand

nf <- length(filenames)
if(nf == 0) {
     print("no files contain any of the search strings")
     return("no files contain any of the search strings")
}

nf
[1] 3

# eliminate duplicate file names
filenames <- unique(filenames)
filenames
[1] "001.csv"    "005txt2csv"
# these are the correct distinct file names for search.strings having
# been set to  c("001", "005", "txt2")

# If got to here, at least 1 file has a character string in search.strings
# in its name, so get the information on these file(s) into a data frame.

################################## get the data frame to be output

# Get the desired output data frame using vectors

dfcolnames <- c("file.name", "modif.date", "size.in.bytes")
# initialize the 3 vectors that will hold this information on the files
# whose names matched all the members of search.strings

fname <- character(0)
fdate <- character(0)
fsize <- numeric(0)

# do, "by hand", the for loop   for(k in 1:nf) {

nf
[1] 3

k <- 1
finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
# get information on the file, includes more info than just the date and the
# most recent time the file was modified
finfo
                  size isdir mode               mtime               ctime
test_dir/001.csv 31271 FALSE  444 2012-09-21 16:00:20 2020-12-13 09:54:03
                               atime exe
test_dir/001.csv 2020-12-27 13:43:46  no

fname <- c(fname, filenames[k])
> fname
[1] "001.csv"

fdate <- c(fdate, as.character(finfo$mtime))
fdate
[1] "2012-09-21 16:00:20"

fsize <- c(finfo$size)
fsize
[1] 31271

# next k in the for loop
k <- 2
finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
finfo
                    size isdir mode               mtime               ctime
test_dir/005txt2csv   11 FALSE  666 2020-12-13 10:03:14 2020-12-13 10:03:14
                                  atime exe
test_dir/005txt2csv 2020-12-27 13:44:49  no

fname <- c(fname, filenames[k])
fname
[1] "001.csv"    "005txt2csv"

fdate <- c(fdate, as.character(finfo$mtime))
fdate
[1] "2012-09-21 16:00:20" "2020-12-13 10:03:14"

fsize <- c(finfo$size)
fsize
[1] 11
# ********  but fsize here should have 2 entries
```

```
#### we see we should have been doing      fsize <- c(fsize, finfo$size)
#### what we did was just have the latest single value for fsize
#### when the data frame was created, R just "recycled" that single value
#### so then all the file sizes were equal the size of the last filename
#### so that accounts for the incorrect file size noted in the test run results
#### but (if you haven't already noted the other mistake) we still need to
#### explain why in the final test case we got a 3rd row consisting of NA's and
#### the file sizes were all NA's

# the for loop for getting information on the files runs from 1 to nf whose value is 3
k <- 3
finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
finfo
            size isdir mode mtime ctime atime  exe
test_dir/NA   NA    NA <NA>  <NA>  <NA>  <NA> <NA>

#### something is obviously wrong here
filenames[k]
[1] NA

#### why is filenames[k] NA when k is 3
#### an obvious question is how many filenames were there after we did
#### filenames <- unique(filenames)
#### there were 2, but we forgot to update nf to be 2

#### so for k equal 3 we got all NA values, and
#### had set fsize <- NA so when fsize got recycled, all
#### the file sizes were NA
```

This illustrates that running the code line by line in the R console can reveal errors or at least focus attention on where to look / what to think about in order to find mistakes.

Here is the corrected version of the function, along with test runs (which now give correct results).

```
search_for_filenames_matching_any_of_the_patterns_and_output_file_info <-
                                          function(directory, search.strings){

# directory is an absolute path (full path) or a path relative to the R working directory to the
# folder to be searched. If want to search the R working directory itself,
# can set directory = "." with the line of code:   directory <- "."
# or could set directory to be the full path to the R working directory.

# search.strings is a character string vector

# Return a data frame of the file names (not including folders) in directory that contain
# ANY of the entries of the search.strings vector somewhere in their file name,
# (or at the beginning of the filename, or at the end of the filename, if so specified).
# The search will be case insensitive (treats lower case and upper case letters as the same).

# The first step is to initialize the filenames vector: filenames <- character(0)

# In a for loop, use R's list.files function to list all the files (file names) matching the
# entries of search.strings, one by one,
# eliminating names of folders, and appending them to filenames
```

```r
# Use the unique function to eliminate duplicates (keep only 1 copy of each file name)

# For the files whose names contain any of the character strings in search.strings, use
# R's file.info function to get the file size and last modification time as in the previous function.

# The output data frame will contain the file size and the last modification time
# for each file that has any member of search.strings somewhere in its file name
# (or at the beginning or at the end of the file name, if so specified)
#
# We will get the file names without the folder path leading to the files included in the name.

# check that search.strings is a non-empty character vector

ns <- length(search.strings)
if(ns < 1) stop("no entries in search.strings")
if(!is.character(search.strings)) stop("search.strings is not a character vector")

# We will return a data frame whose first column contains the files in directory
# that contain any character string in the search.strings vector somewhere in their name.
# The second column will contain the last time (and date) the file was modified,
# and the third column will be the file size (in bytes).

# The first step is to initialize the filenames vector
filenames <- character(0)

# Then in a for loop, for each entry S of search strings,
# use list.files to get the vector V the file names in directory that
# contain S in their file name and append V to filenames (after eliminating any folder names).
# do not include the path to the file in the file name.

# To eliminate names of any folders (directories) that are in V:
# do paste(directory, "/", V, sep = "") to get the filenames including either
# the relative path from the R working directory or the absolute path (depending on what
# directory is); use these names in the R dir.exists function to check for folder
# (directory) names in filenames

   for (k in 1:ns) {
      V <- list.files(directory, pattern = search.strings[k],
                      full.names = FALSE, ignore.case = TRUE)
#     exclude directory names from V (we need to do this since we are "adding"
#     the file names in V to filenames and want only file names, not folder names
#     if V is empty (character(0)) then skip this
      if(length(V) > 0) V <- V[!dir.exists(paste(directory, "/", V, sep = ""))]
      filenames <- c(filenames, V)
   }

### It is important to note we could have also "run the for loop" in this fashion:
### for (S in search.strings) {
###    V <- list.files(directory, pattern = S,
##############   rest of the for loop
###

nf <- length(filenames)
if(nf == 0) {
   print("no files contain any of the search strings")
   return("no files contain any of the search strings")
}

filenames <- unique(filenames)
nf <- length(filenames)  # need to do this since may have eliminated some duplicate name(s)

# If got to here, at least 1 file has a character string in search.strings
# in its name, so get the information on these file(s) into a data frame.

################################## get the data frame to be output

# Get the desired output data frame using vectors

dfcolnames <- c("file.name", "modif.date", "size.in.bytes")
# initialize the 3 vectors that will hold this information on the files
# whose names matched any of the members of search.strings

fname <- character(0)
fdate <- character(0)
fsize <- numeric(0)

for(k in 1:nf) {
    finfo <- file.info(paste(directory, "/", filenames[k], sep = ""))
# needed to include the path to the file so file.info can locate it
    fname <- c(fname, filenames[k])
    fdate <- c(fdate, as.character(finfo$mtime))
    fsize <- c(fsize, finfo$size)
}

df <- data.frame(fname, fdate, fsize, stringsAsFactors = FALSE)
colnames(df) <- dfcolnames

################################## finished getting the data frame to be output

# Write the data frame out to a tab delimited text file called scrlisting.txt in directory
# (i.e., in the folder specified by the argument directory this function was called with).

outpfilename <- paste(directory, "/", "scrlisting.txt", sep = "")
# One can rename this "scratch file" as desired after viewing it (best viewed in Excel or equivalent).
write.table(df, file = outpfilename,
            append = FALSE, quote = FALSE, sep = "\t",
            row.names = FALSE, col.names = TRUE)
# This call to write.table will write out a data frame
# as one would usually want; it specifies the column separator to be a tab

return(df)
}

# do test runs on your computer
```

Here are the results of the test runs using the correct version of the code above (the results match the file names and file information in the test_dir folder in my computer).

```r
# test runs on my computer
directory <- "C:/berger/R_course_cs/test_dir"  # an absolute path to test_dir

# note if, in RStudio, you click on knit to invoke knitr on a .Rmd file,
# it will run R code with the folder the file is in as the R working directory!!
# using absolute paths may avoid this issue

# note the  scrlisting.txt  file contains the output from
# the previous run of the search function so its modification date and file size will vary
list.files(directory)
```

```
##  [1] "001.csv"         "002.csv"         "003.txt"         "004csvfile.txt"
##  [5] "005txt2csv"      "1.csv"           "308.csv"         "folder001txtcsv"
##  [9] "scrlisting.txt"  "txt.csv"
```

```r
search.strings <- c("001", "30")  # should NOT get the folder name folder001txtcsv
search_for_filenames_matching_any_of_the_patterns_and_output_file_info(directory, search.strings)
```

```
##   file.name          modif.date size.in.bytes
## 1   001.csv 2012-09-21 16:00:20         31271
## 2   308.csv 2012-09-21 16:00:20         74305
```

```r
search.strings <- c("1234", "2222")  # test no matches
search_for_filenames_matching_any_of_the_patterns_and_output_file_info(directory, search.strings)
```

```
## [1] "no files contain any of the search strings"
```

```
## [1] "no files contain any of the search strings"
```

```r
search.strings <- c("001", "005", "txt2")  # test use of unique
search_for_filenames_matching_any_of_the_patterns_and_output_file_info(directory, search.strings)
```

```
##    file.name          modif.date size.in.bytes
## 1    001.csv 2012-09-21 16:00:20         31271
## 2 005txt2csv 2020-12-13 10:03:14            11
```

Hope this sequence of exercises was informative and good practice. The next exercise will be composing a function to find files names that do NOT match any of the entries of search.strings This will be more an exercise in using basic constructs in R, and then an
introduction to functions in R that carry out operations with sets, than a function with practical use.

= =