a single function version of makeVector and cachemean from the JHU Data Science Specialization on Coursera R Programming course

Alan E. Berger January 24, 2025

available at https://github.com/AlanBerger/R_Programming_GitHub_files

Motivation

The R Programming course in the Johns Hopkins University Data Science Specialization on Coursera (a 10 course sequence) covers many topics in the R Programming language, including ***lexical scoping***. As part of a programming exercise involving lexical scoping, the pair of R functions, **makeVector** and **cachemean**, are made available to the class; they are publicly available within the **README.md** file in Professor Roger D. Peng's GitHub site: https://github.com/rdpeng/ProgrammingAssignment2 A detailed description of how makeVector_and_cachemean work is available in an article by Leonard Greski at: https://github.com/lgreski/datasciencectacontent/blob/master/markdown/rprog-breakingDownMakeVector.md

If you download a copy of this pdf file into your computer, then the web links in it will work.

A call to the makeVector function returns an R object that is a list of 4 functions, and also is a ***function closure***, well described in Leonard Greski's article, but not referred to by that name. Function closures are also discussed in, for example, https://github.com/AlanBerger/R_Programming_GitHub_files/blob/master/reply-to-where-function-closure-variables-are-8-Jan-2020.md

A good observation in the Discussion Forum in the JHU R Programming class noted that someone could "misuse" the **setmean** function provided by makeVector, replacing the current saved (*cached*) value of the mean of a given vector with an incorrect value.
The programming exercise involving makeVector and cachemean was focused on understanding lexical scoping, and not concerned at that point with producing "library quality" code that would "shield" the user from misadventure. However, I thought it an interesting exercise to write a single function that would retain the primary functionality of makeVector and cachemean, illustrate use of lexical scoping and the properties of a function closure, be simpler to use, and perhaps be a more concise demonstration of key concepts.

This function, **get_and_cache_mean**, is provided below, along with sample runs. It is heavily commented, and uses long (one might say overly long) variable names to make it easier to follow.

The get_and_cache_mean function and example runs

The **get_and_cache_mean** function has no argument. It codes for and returns the function **get_the_mean_of_v**

Since get_and_cache_mean defines and returns the function get_the_mean_of_v, get_the_mean_of_v is also a **function closure** and has the capability of "retaining" variables in its parent environment, and accessing and modifying them (see the referenced articles above, or other sources on function closure). When get_the_mean_of_v is called, its argument should be a non-empty numeric vector, call it v. get_the_mean_of_v calculates and returns the mean of v, and "stores" the calculated mean. The key point of this programming example is to use lexical scoping and the properties of a function closure so that if get_the_mean_of_v has been called before in the current R session, and the most recent call was with the same value of v as the present call, then get_the_mean_of_v will "fetch" the previously calculated value of the mean that was stored in the parent environment of get_the_mean_of_v (possible since get_the_mean_of_v is a function closure),
rather than recalculing the mean.

If want to copy code, copy it from the Rmd (R Markdown) file, not the pdf file (copying code from a pdf file in GitHub might entrain formatting characters unacceptable to R).

```
get_and_cache_mean <- function() {
# call it via:     get_the_mean_of_v <- get_and_cache_mean()

# get_and_cache_mean returns the function get_the_mean_of_v

# get_the_mean_of_v  will have 1 argument,  v (a numeric vector),
# and will calculate the mean of v,
# and  get_the_mean_of_v  will also be a function closure

    m <- NULL  # initialize the mean m of v to be NULL so
#                        know need to calculate it

    v_for_which_calculated_the_mean <- NULL
#  the v for which have calculated the mean, initially NULL

# define the function to be returned by  get_and_cache_mean

    get_the_mean_of_v <- function(v = numeric()) {
#       calculate the mean of v, or when appropriate fetch and
#       return the cached (saved) mean

# do some testing for whether the value that  get_the_mean_of_v
# was called with is "admissible" for what  get_the_mean_of_v
# is designed to do

       if(is.numeric(v) == FALSE)
            stop("the argument v of  get_the_mean_of_v  is not numeric")
       if(is.vector(v)  == FALSE)
```

```
            stop("the argument v of get_the_mean_of_v is not a vector")
       if(length(v) < 1) stop("length of v is < 1")
#      note a variable that is a single value is, in R,
#      a vector of length 1
#      a matrix and a data frame are examples of R objects
#      that are not vectors

        m_from_parent_envir <- m
#  use R's lexical scoping to acquire m

        v_from_parent_environment <-  v_for_which_calculated_the_mean
#       use R's lexical scoping to acquire
#       v_for_which_calculated_the_mean

#       test whether there is a cached (saved) mean that is
#       appropriate to use

#       need to be VERY careful with any NULL values
#       use the  is.null   function to test for a potential NULL value;
#       if have verified that
#       the value is not NULL,
#       then can do regular logical tests or arithmetic
#       on it (the length of the NULL R object is 0)
#
   if(!is.null(m_from_parent_envir) && !is.null(v_from_parent_environment)) {
#       if got to here, neither is NULL, so have cached a mean
#
#       test whether the value of v used in the call to  get_the_mean_of_v
#       is the one for which the cached mean has been calculated

            if(identical(v_from_parent_environment, v)) {
#           note for example c(1, 2, 3) is numeric while 1:3 is
#           the vector of integers c(1L, 2L, 3L) so they will not be
#           identical R objects
#          (even though their entries are equal numbers)

#          if pass the if test above, can use the cached (saved) mean

                message("getting cached value of the mean of v")
                return(m_from_parent_envir)  # returns the cached mean
#               and exits the get_the_mean_of_v function

            }   # closing brace for the if test immediately above
       }   # closing brace for the if test above that checks for NULL values
```

```
#       if got to here, need to calculate the mean of v
#       which is the v that is the argument of the current call
#       of  get_the_mean_of_v

        v_for_which_calculated_the_mean <<-  v
#                 use the superassignment operator <<- to
#                 update   v_for_which_calculated_the_mean
#                 in the parent environment of   get_the_mean_of_v

#                 the parent environment is a preserved version of
#                 the "execution time" environment of
#                 get_and_cache_mean

#                 The R statement
#                 get_the_mean_of_v <- get_and_cache_mean()
#                 defines and returns the function    get_the_mean_of_v
#                 and hence  get_the_mean_of_v
#                 is also a "function closure"

#                 As a crucial consequence of  get_the_mean_of_v  being a
#                 function closure, the variables (R objects) m
#                 and  v_for_which_calculated_the_mean
#                 defined in the original call to  get_and_cache_mean
#                 are preserved and available within the
#                 parent environment of get_the_mean_of_v
#                 so available, if properly accessed, to
#                 the function  get_the_mean_of_v

        mean_of_v <- mean(v)    # note if v were v = numeric(),
#                               i.e., a numeric vector of length 0,
#                               its mean would be NaN
        m <<-  mean_of_v  # use the superassignment operator <<-
#                 to cache the mean of v in the parent
#                 environment of   get_the_mean_of_v

        message("returning the calculated mean of v, NOT a cached value")
        message("have now cached the mean of the current value of v")
        return(mean_of_v)
    }  # ending brace for the function   get_the_mean_of_v

}  # ending brace for the function  get_and_cache_mean

####################################
```

```r
# Now do some sample runs

get_the_mean_of_v <- get_and_cache_mean()
# get_the_mean_of_v   is a function and also is a function closure

v <- c(6, 7, 8)  # simple example

get_the_mean_of_v(v)
```

```
## returning the calculated mean of v, NOT a cached value

## have now cached the mean of the current value of v

## [1] 7
```

```r
#  check that the mean was cached in the previous
#  call of  get_the_mean_of_v
get_the_mean_of_v(v)
```

```
## getting cached value of the mean of v

## [1] 7
```

```r
new_v <- c(1, 2, 3)
get_the_mean_of_v(new_v)   # check works with new vector
```

```
## returning the calculated mean of v, NOT a cached value
## have now cached the mean of the current value of v

## [1] 2
```

```r
get_the_mean_of_v(new_v)   # again, check cached the mean
```

```
## getting cached value of the mean of v

## [1] 2
```

```r
#          note c(1, 2, 3) is numeric while 1:3 is
#          the vector of integers c(1L, 2L, 3L) so they will not be
#          identical R objects, and so
#          get_the_mean_of_v will recalculate the mean in this case:
get_the_mean_of_v(1:3)
```

```
## returning the calculated mean of v, NOT a cached value
## have now cached the mean of the current value of v

## [1] 2
```

```r
get_the_mean_of_v(1:3)  # again, check cached the mean
```

```
## getting cached value of the mean of v

## [1] 2
```

```r
## the tests below will give errors and put the R session into Browse
## (note they stop R Markdown, so not run)
## exit Browse by hitting the Escape key

## get_the_mean_of_v("character string")
# check non-numeric argument gets error

## matr <- matrix(1:12, 3, 4)  # 3 row by 4 column matrix
## get_the_mean_of_v(matr)
# check argument that is not a vector gets error

## get_the_mean_of_v(numeric(0))
# check argument that has length < 1 gets error
```

Hope this is helpful in understanding R's lexical scoping and how a function closure works.

Note the corresponding version of the code above for the inverse of a matrix is NOT what is required for the second assignment in the R Programming course.

= =