Discussion Forums

Week 3

Week 3

[TIPS] Demystifying makeVector()

Leonard Greski Mentor Week 3 · 4 months ago

Hello everyone. I finally got the time to sit down and write this article. After lots of feedback from students expressing frustration about the challenges in understanding this assignment, I've written Demystifying makeVector() to explain the key R language features that are used within this function.

Consider it the missing lecture on lexical scoping within R functions...

regards,

Len

31 Upvote · Follow 4 · Reply to Leonard Greski

Earliest **Top** **Most Recent**

JG Jesus Garcia · 2 days ago

Another quick clarification

When using

setmean <- function(mean) m <<- mean

we are not calling the actual mean function, correct? "mean" here is used as a variable name that is holding the mean calculated in cachemean()?

0 Upvote · Reply

JG Jesus Garcia · 2 days ago

quick question

What is the purpose of the "further arguments to pass" (. . .) in cachemean() ?

I'm talking about cachemean <- function(x, **...**) {

and m <- mean(data, **...**)

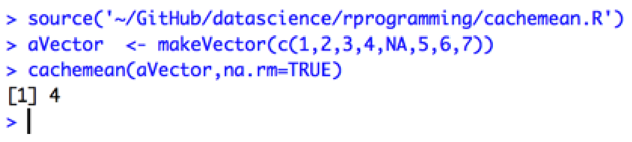
0 Upvote · Hide 1 Reply

Leonard Greski Mentor · 16 hours ago · Edited

Hello Jesus.

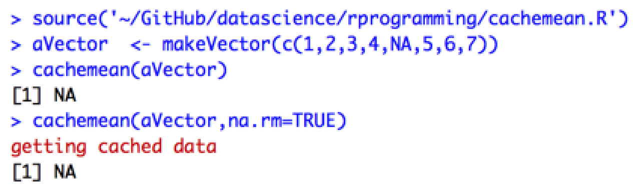
Help Center

mean() includes optional arguments such as **na.rm** to remove missing values from the mean calculation. The ellipsis for "other arguments to pass" allows one to call **cachemean()** as **cachemean(aVector,na.rm=TRUE**).



```
> source('~/GitHub/datascience/rprogramming/cachemean.R')
> aVector  <- makeVector(c(1,2,3,4,NA,5,6,7))
> cachemean(aVector,na.rm=TRUE)
[1] 4
> |
```

Note that support for the ellipsis highlights a subtle defect in the sample code related to missing value handling. If you create a vector that includes **NA** values and execute **cachemean()** without the **na.rm=TRUE** parameter, it will cache **NA** and subsequent calls will return **NA** regardless of whether they include the **na.rm=TRUE** argument.

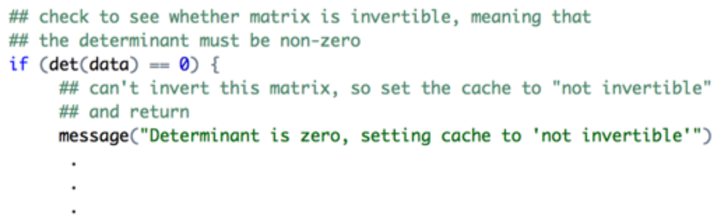


```
> source('~/GitHub/datascience/rprogramming/cachemean.R')
> aVector  <- makeVector(c(1,2,3,4,NA,5,6,7))
> cachemean(aVector)
[1] NA
> cachemean(aVector,na.rm=TRUE)
getting cached data
[1] NA
```

Given that the main point of the programming assignment is to help students learn the concept of lexical scoping, it's understandable that the professor didn't code it to handle situations where the result of **makeVector()** contains missing values. The extra code would distract students' attention from the main point of the lesson.

In a similar vein, **cachematrix()** can be coded in a way that includes a check to handle non-invertible matrices, but the instructions specifically state to assume the matrix that is input to **makeCacheMatrix()** is invertible.

Since the calculation of the inverse of a matrix includes division by its determinant, if the determinant of a matrix is zero it is non-invertible, also known as "singular."



```
## check to see whether matrix is invertible, meaning that
## the determinant must be non-zero
if (det(data) == 0) {
     ## can't invert this matrix, so set the cache to "not invertible"
     ## and return
     message("Determinant is zero, setting cache to 'not invertible'")
      .
      .
      .
```

regards,

Len

1 Upvote

Reply

Reply

CW

Claus Walter · 6 days ago

Hi Leonard

Many thanks for the article! I tried Google and some books first to understand the sample functions, but it was really frustrating. But with your help, no problem! I think I now understand how it works. Very different from other

Help Center

languages.

0 Upvote · Reply

Mohamed Elfatih Khalil Ali · 9 days ago

Hello everyone, please review my week 3 assignment

http://github.com/MohamedElfatihKhalil/Peer-graded-Assignment

0 Upvote · Reply

YW

Yunfei Wu · 20 days ago

Thank you very much for detailed explanation, lots of lines make sense now, very interesting!

0 Upvote · Reply

Maarten Klein · a month ago · Edited

Hi Leonard, I was wondering why the use two functions (makeVector and cachemean) and not one function.

```
cacheVector <- function(x = numeric()) {
    m <- NULL
    set <- function(y) {
        x <<- y
        m <<- NULL
    }
    get <- function() x
    mean <- function(...) {
        if (!is.null(m)) {
            message("getting cached data")
            return(m)
        }

        m <<- base::mean(x, ...)
        m
    }
    list(set = set, get = get, mean = mean)
}


> m <- cacheVector(1:100)
> m$mean()
[1] 50.5
> m$mean()
getting cached data
[1] 50.5
>
```

By this you don't have the problem that the cachemean can be called with a vector instead of a makeVector object and that the getmean() and setmean() functions which now are exposed to every one (instead only to cachemean) are not necessary. Using het getmean() function directly can have unexpected result.

0 Upvote · Hide 1 Reply

Leonard Greski Mentor · 16 hours ago

Hello Maarten. I agree that your approach is a valid alternative. I don't have visibility to Professor Peng's reasoning behind the second function other than the fact that **makeVector()** is designed as a "data object" that contains only get and set type functions for its two attributes, **x** and **m**.

regards,

Len

0 Upvote

Reply

Help Center

Reply

HQ

HECTOR QUIROZ · a month ago · Edited

Hi Len,

There's one thing that is not clear to me yet. Could you please explain me how does the setmeal function in the makeVector environment works?

setmean <- function(mean) m <<- mean

Is 'mean' just a variable within the environment of setmean that is passed to m in the local environment of makeVector? Your help will be much appreciated.

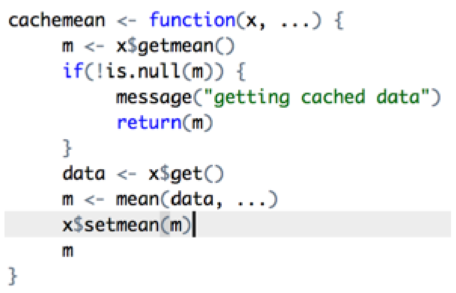
Thanks,

Héctor.

0 Upvote · Hide 2 Replies

Leonard Greski Mentor · a month ago

No, **mean** is the argument to the **setmean()** function. If you look at the code for **cachemean()**, you'll see how it's used to set the value of **m** in the **makeVector()** object:



```
cachemean <- function(x, ...) {
     m <- x$getmean()
     if(!is.null(m)) {
          message("getting cached data")
          return(m)
     }
     data <- x$get()
     m <- mean(data, ...)
     x$setmean(m)
     m
}
```

regards,

Len

0 Upvote

HQ

HECTOR QUIROZ · a month ago

I got it, thanks!

0 Upvote

Reply

Reply

Igor Calado · a month ago

I think I understand the logic and the rules of scoping and assigning and yet my code is not acting like I expected. Once it sets m to NULL in the second line, the rest of the code seems to make no difference: at the end, m is always null. If I assign it the string "Bang you're wrong", it will stick to that character value no matter what it is assigned in set() and setmean(), and return "Bang you're wrong" if I ask it to print(m) at the end.

Help Center

From what I had studied, the <<- operator, as in m <<- 900 (line 5) should prompt R to search for the value of the m element in the parent evironment - it would then find m <- "Bang you're wrong" in the second line and reset it to 900. But it's not working! I tried with <-, <<-, = and == in the 5th line but is does not seem to affect m. The same for line 8, where the setmean function is.

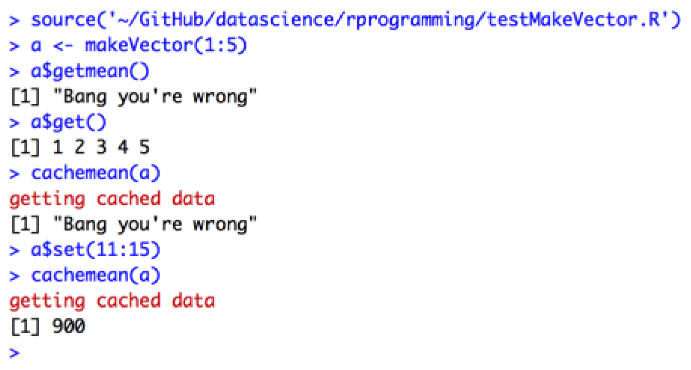
Any help would be much appreciated.

```
 1  makeVector <- function(x = numeric()) {
 2    m <- "Bang you're wrong"
 3    set <- function(y) {
 4      x <<- y
 5      m <<- 900
 6    }
 7    get <- function() x
 8    setmean <- function(mean) m <<- mean
 9    getmean <- function() m
10    list(set = set, get = get,
11         setmean = setmean,
12         getmean = getmean)
13    print(m)
14  }
```

0 Upvote · Hide 3 Replies

Leonard Greski Mentor · a month ago

Hello Igor. The last line, **print(m)** breaks the function. Why? **makeVector()** should return the **list()** object, not **m**. If you deleted the last line, add **cachemean()**, and run the function, you get the following:



```
> source('~/GitHub/datascience/rprogramming/testMakeVector.R')
> a <- makeVector(1:5)
> a$getmean()
[1] "Bang you're wrong"
> a$get()
[1] 1 2 3 4 5
> cachemean(a)
getting cached data
[1] "Bang you're wrong"
> a$set(11:15)
> cachemean(a)
getting cached data
[1] 900
>
```

Since **m** is never NULL, **cachemean()** always retrieves a cached value, but you can see that it does change from "Bang you're wrong" when we reset the value of **x** with **a$set(11:15)**.

regards,

Len

1 Upvote

Igor Calado · a month ago

It works!!!! Thank you so much Len!

But now I have another, simpler question: I thought the cache was calculated the first time inside the makeVector function and stored in m. After, in the cachevector function, it would check if it was empty and calculate again if it is true.

But from this exemple and others I've seen you give in this thread, it seems that the makeVector function only *creates* the special object, which is a vector with an empty space where we will store someday the cache. Then we would apply the cachevector function and it would calculate and insert the cache. Hence, the cache is only retrieved after you run the cachevector function for the first time.

Is that it? That would explain why I thought everything was wrong: I was expecting to cache the value of the mean (or inverse matrix) when running the first function.

Thanks once more.

0 Upvote

Leonard Greski Mentor · a month ago

That's correct. **cachemean()** either retrieves the mean from the cache, or executes the **setmean()** function on the object passed to it.

regards,

Len

0 Upvote

Reply

Reply

Rishi Jain · a month ago

Hi Len,

Thanks a **ton** for creating the demystifying section, wouldn't have been able to understand without the document. Kudos good work. The 2 question examples in the last helped the most!!

I have got a query related to Q2 - Why is set() never used in code. I understood your explanation on a$set(c(30,50,90,120,180)) would help in directly setting variable x in set. However, in case I try to set a$setmean(10) with a value of 10. And then try to use a$getmean() it still gives me NULL. Should it give a 10 as we have "setmean <- function(mean) m <<- mean" and in this m is updated in parent.

Best regards,

0 Upvote · Hide 1 Reply

Leonard Greski Mentor · a month ago

Hello Rishi. **set()** isn't used in the code because it is unnecessary unless one has already instantiated an object of type **makeVector()** and is setting the value of the input vector for a second time.

Also, **setmean()** works correctly for me:

```
> source('~/GitHub/datascience/rprogramming/cachemean.R')
> a <- makeVector(1:5)
> cachemean(a)
[1] 3
> a$getmean()
[1] 3
> a$set(10:20)
> a$get()
 [1] 10 11 12 13 14 15 16 17 18 19 20
> a$setmean(mean(a$get()))
> a$getmean()
[1] 15
>
```

regards,

Len

0 Upvote

Help Center

Reply

Reply

DR

David Steven Rabiger · 2 months ago

Why set()?

This function seems unnecessary in makeVector() and in fact deleting set() from makeVector() still produces the expected output. I understand that including set() allows you to change the value of x without initializing a new instance of makeVector(), but why is that desirable? Either way, a new mean must be computed.

Why is:

> a$set(1:5)

preferable to:

> a <- makeVector(1:5)

2 Upvote · Hide 4 Replies

Leonard Greski Mentor · a month ago

Once **a** has been loaded into memory, **makeVector()** isn't necessary to manipulate **a**. In fact, calling **makeVector()** a second time creates a new object, which is more expensive than reusing the one that was previously created.

regards,

Len

1 Upvote

BALSHER SINGH · a month ago · Edited

a$set(1:5) will save computing cost and change m to NULL. Yeah?

Thus use it to save computing time and when you call cachemean() on ,a, again (!is.null(m)) is FALSE.

0 Upvote

Leonard Greski Mentor · a month ago

Yes. Once an object of type **makeVector()** is initialized, one uses **set()** to reset the value of **x**, so **cachemean()** can set or access the cached value of **m**.

Accessing the values via getters and setters is how object orientation works in the S3 object system, as I describe in Object Oriented Programming and R.

regards,

Len

0 Upvote

BALSHER SINGH · a month ago

Thank you thank you. (Object Orientation) S3 and its behavior and state, these are new concepts to me.

Help Center

One things that does bug me is that only way I got going with this assignment was after I saw your Demystifying MakeVector post in the discussion forum (I have not had time to read it), while glossing over it I saw your code at the bottom and realized how the two functions work together, namely that output of makemean() is the input for cachemean(). Then I could start to put the puzzle together.

Now, you know the subtleties of object system, etc, but I didn't know what that heck is going on when I first saw the two functions.

At present I am seriously strapped for time and do the best I can to get though as many daily learning objects before bed time. But I am seriously considering allotting a part of my daily schedule to read Software for data analysis by John Chambers. I don't want to use it as a reference but would like to read it cover to cover like a tradition text.

Good idea? Will it help to deepen my grasp as I divulge into R.

Thanks a ton.

0 Upvote

Reply

Reply

R

Rui_Lian · 2 months ago

Hi, Len,

two questions for makeVector():

1. Why are the hexadecimal number returned? Is this number the "physical position" of the function environment in the RAM?

2. Why do We use "void" functions w/o any arguments in "get" and "getmeans"? Why can't we just use 'get <- x', 'getmean<-mean'?

Maybe these are stupid questions, but I don't have any programming experience before.

Thanks in advance!

Rui

1 Upvote · Hide 2 Replies

Help Center

Leonard Greski Mentor · 2 months ago

Hello Rui. Regarding your questions...

1. Yes, the hexadecimal represents a memory location.
2. The functions without arguments are required to access x and m because they are not directly accessible outside the makeVector() closure / environment. Since the output from makeVector() is an object of type list, we use the $ form of the extract operator to access the functions in the list, and these functions have access to x and m.

regards,

Len

1 Upvote

R

Rui_Lian · 2 months ago

Thanks, Len.

I'm more comfortable now: functions without arguments are siblings, whose functionalities are transient but the RAM address is kept for 2nd function to retrieve if the data is not updated. That will reduce the calculation and bypass the CPU-RAM talk. this could be very critical for big data.

0 Upvote

Reply

Reply

Ram ji dubey · 2 months ago

Thanks for this Article, though i submitted my assignment but now i can think of doing it differently. again thanks for your efforts.

Thanks & Regards

Ram

0 Upvote · Reply

Tomasz Grzegorzek · 2 months ago

Hello everyone. When it comes to programming I'm a newbie but somehow I got to OOP because of 'setters and getters' method. I kept reading, asking question, searching for an answer and I got really deep to the point where I basically tried to learn OOP in Java. I know that this assignment is basically about lexical scoping but please verify my way of thinking about these two function **makeVector** and **cachemean.** Is it a good comparison to treat makeVector as a class in OOP and cachemean as a way of maintaing objects? That is, makeVector specifies the boundaries of the object: it can set value, give the value to another object, get mean and give it to another object and of course, on every run it creates new vector. Cachemean, on the other hand, comunicates with makeVector and check what it can do at a moment, then return value (mean). Is it close to the true? I can't sleep. Any suggestion for understanding the flow of these function will be appreciated :) Thank you.

0 Upvote · Hide 1 Reply

Leonard Greski Mentor · 2 months ago · Edited

Hello Tomasz.

Help Center

Generally speaking, you've got it right.

makeVector() is like a constructor class if you're using object oriented jargon. cachemean() then operates on objects of type makeVector().

Take a look at my articles makeCacheMatrix() as an Object, and R objects, S Objects, and Lexical Scoping for more background on how R implements Object orientation.

I apologize for the delayed response. I've been in India this week and haven't have reliable internet access until I returned to the airport to fly back to the United States in a few hours.

regards,

Len

1 Upvote

Reply

Reply

Barrett smith · 4 months ago

Leonard, thank you for putting this together. I had groked the "what" and "how" of makeVector but your write up helped me finally get the "why": that using lexical scoping is how R handles cases which other languages would solve using developer defined object classes.

0 Upvote · Reply

MG

Manish Gyawali · 4 months ago

Is the solution to the matrix inversion problem significantly different from that of the means question? My understanding is that they should be quite similar with the difference being in how one reads the matrix and how the matrix is solved. So if I was to essentially copy the means program and make a few changes, I should be okay. However, I began to wonder if a matrix was essentially a different type of data-type and whether you needed to deal with it differently.

Having read Leonard's article a couple of times, I was quite sure I'd understood everything I needed to as far as scoping is concerned. So my strategy for the problem was:

1. makeCacheMatrix`: This function creates a special "matrix" object that can cache its inverse.

To make a cache Matrix, I reasoned that I would need to initialize the objects, define the behaviors for the objects created by make_cache_Matrix, and finally create a new object by returning a list.

2.cacheSolve`: This function computes the inverse of the special "matrix" returned by `makeCacheMatrix` above. If the inverse has already been calculated (and the matrix has not changed), then the`cachesolve` should retrieve the inverse from the cache.

I reasoned that this is to be done similarly to that of the means problem.

However, when I run the program, I encounter a problem. So I'm wondering if I'm on the right track. Again, is there anything specifically about matrix data types that makes them different to work with than normal vectors?

Help Center

0 Upvote · Hide 2 Replies

Alan E. Berger · 4 months ago · Edited

First off, as noted in the "asking questions" lecture, it improves your chances of getting helpful advice if you describe in detail what the "problem" you encountered was: did you get an error message (what was it)? did you get a result that was not the matrix inverse of the matrix you submitted? (what was the matrix you submitted, and the matrix that got returned that should have been its matrix inverse) etc.

Since this relates to a graded assignment there are limits to how much information can be provided by other than one of the mentors who have had training on where the line is on offering advice on homework problems -- but I think the following might be helpful orientation (while does not provide a solution to the assignment).

From the instructions on the assignment, and the many comments in this thread: the makeVector function "stores" the vector that is "given to it" (in its argument) in the variable x in the environment of makeVector and initializes the mean of x (to be placed in the variable m) to be NULL (m is NULL until the value of the mean of x is calculated and assigned to m), and defines the 4 functions set (argument y), get (empty argument) , setmean (argument mean) , getmean (empty argument). These 4 functions are "explicitly" returned by makeVector (in a list) while (see all the essential to read "articles" and comments by Leonard Greski) x and m are also available in the environment of makeVector, and the functions use the scoping properties of R.

That is, if one invokes makeVector by

makeVector_object <- makeVector(some_vector)

then makeVector_object is a list containing the 4 functions; accessible as

makeVector_object$set

makeVector_object$get

makeVector_object$setmean

makeVector_object$getmean

and also

makeVector_object

has x (whose value equals the argument some_vector) and m in its environment, which is available to

cachemean when it is invoked by:

cachemean(makeVector_object)

The assignment is to write a pair of R programs which do the analogous thing, this time the first program is "fed" a square invertible matrix m1 instead of a vector, and the second program is to return (and "cache") the matrix inverse of m1 (call it m1_inverse);

m1_inverse is to be obtained, per assignment instructions, using the R function solve (available in the basic R installation).

So if these 2 functions are written correctly, then (using a simple example matrix m1, and its matrix inverse) the R commands below obtain and cache the matrix inverse of m1:

Help Center

m1 <- matrix(c(1/2, -1/4, -1, 3/4), nrow = 2, ncol = 2)

# a simple 2 by 2 invertible matrix with an even simpler looking inverse

myMatrix_object <- makeCacheMatrix(m1)

cacheSolve(myMatrix_object)

# returns (and caches) the matrix inverse of m1

# which is equal to

# matrix(c(6,2, 8,4), nrow = 2, ncol = 2)

# check m1 %*% matrix(c(6,2, 8,4), nrow = 2, ncol = 2)

# gives the 2 by 2 identity matrix matrix(c(1,0, 0,1), nrow = 2, ncol = 2)

doing

cacheSolve(myMatrix_object)

a second time should simply fetch (but not recalculate) the inverse of m1 (and report that it only fetched it)

0 Upvote

Leonard Greski Mentor · 4 months ago

Hello Manish. I thought I had responded to your post yesterday, but it seems that my response didn't get saved to the Discussion Forum.

The logic of using makeVector() and cachemean() as the basis for your assignment is reasonable. I was able to convert the example functions into a working version of the programming assignment, and test the converted functions in less than 2 minutes. It's pretty easy to generate a working version of the assignment that meets most of the criteria in the grading rubric in this manner.

However, the most important part of the assignment is the comments. These illustrate whether you understand what the code is doing. Please take the time to comment more than just he overall function. Explain why the programming statements work the way they do, and you'll get the most benefit out of the assignment.

As Alan noted in his post, to help you debug the functions, we'll need a more detailed description of the error you're encountering.

regards,

Len

1 Upvote

Reply

Reply

Help Center

1 2 3

Reply

Reply

Help Center