Lexical scoping, also known as static scoping, is a convention used with many modern [programming languages](https://www.techtarget.com/searchcloudcomputing/tip/11-cloud-programming-languages-developers-need-to-know). It refers to setting the *scope*, or range of functionality, of a [variable](https://www.techtarget.com/whatis/definition/variable) so that it may be called (referenced) from within the block of [code](https://www.techtarget.com/whatis/definition/code) in which it is defined. The scope is determined when the code is compiled. A variable declared in this fashion is sometimes called a *private variable*.

In a programming language, scope refers to the area where a [function](https://www.techtarget.com/whatis/definition/function) or variable is visible and accessible to other code. Below are some common terms associated with scope:

* *Global scope* refers to a global or public space.
* *Local scope* refers to a local or restricted region.
* *Scope chain* refers to the unique spaces that exist from the scope where a variable was called to the global scope.

**Understanding lexical scoping**

In a programming language, an item's lexical scope is the place in which it was created. The scope of the variable is determined by the [program's](https://www.techtarget.com/searchsoftwarequality/definition/program) textual (lexical) structure. Variables can be declared within a specific scope and are only accessible within that region.

In other words, lexical scope refers to the ability of a function scope to access variables from the parent scope. When there is lexical scope, the innermost, inner and outermost functions may access all variables from their parent scopes all the way up to the global scope. However, no scope may access the variables from the functions defined inside it. Thus, the child function is lexically bound to the parent function.

The first function, makeVector creates a special “vector”, which is really a list containing a function to

set the value of the vector

get the value of the vector

set the value of the mean

get the value of the mean

makeVector <- function(x = numeric()) {

m <- NULL

set <- function(y) {

x <<- y

m <<- NULL

}

get <- function() x

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

getmean <- function() m

list(set = set, get = get,

setmean = setmean,

getmean = getmean)

}

The following function calculates the mean of the special “vector” created with the above function. However, it first checks to see if the mean has already been calculated. If so, it gets the mean from the cache and skips the computation. Otherwise, it calculates the mean of the data and sets the value of the mean in the cache via the setmean function

cachemean <- function(x, ...) {

m <- x$getmean()

if(!is.null(m)) {

message("getting cached data")

return(m)

makeCacheMatrix <- function(x = matrix()) {

j <- NULL

set <- function(y){

x <<- y

j <<- NULL

}

get <- function()x

setInverse <- function(inverse) j <<- inverse

getInverse <- function() j

list(set = set, get = get,

setInverse = setInverse,

getInverse = getInverse)

}

##Please include your own comment to explain your code (Required in Rubric)

cacheSolve <- function(x, ...) {

## Return a matrix that is the inverse of 'x'

j <- x$getInverse()

if(!is.null(j)){

message("getting cached data")

return(j)

}

mat <- x$get()

j <- solve(mat,...)

x$setInverse(j)

  j

}data <- x$get()

m <- mean(data, ...)

x$setmean(m)

        m

}